**Documentación React Paso a Paso**

**Presentando JSX**

Sintaxis como ejemplo:

* const element = <h1>Hello, World!</h1>;

Esta sintaxis no es ni un string ni un HTML, se le denomina JSX y es una extensión de la sintaxis de Javascrip se recomienda usar con React para describir mejor cómo debería ser la interfaz de usuario.

**¿Por qué JSX ?**

React acepta el hecho de que la lógica de renderizado ( JS ) está intrínsicamente unida a la lógica de la interfaz de usuario ( HTML AND CSS ).

React en lugar de separar artificialmente las tegnologías en una maquetación donde separa HTML, CSS y JS , React, separa intereses ( fragmentos de algo ) y estos son acoplados en diferentes componentes, osea, de algo grande lo divide en pequeñas cosas y cada una de estas son componentes.

Ejemplo:

* Const nombre = ‘Juan Alcala’;
* Const elemento = <h1>Mi nombre es : {nombre}</h1>

En **JSX** puedes agregar cualquier código javascript dentro de llaves, como se muestra en el ejemplo donde se va a pintar la variable constante de nombre. También este tipo de Sintaxis puede ser llamado dentro de funciones, condicionales o bucles de Javascript.

**Especificando Atributos con JSX**

Se pueden usar comillas para especificar strings literales como atributos:

* Const element = <div tabIndex=”0”></div>;

También se puedne usar llaves para insertar una expresión Javascript en un atributo:

* Const = element = <img src={ user.avatarUrl } ></img>;

\***Recuerda se debe usar comillas para valores strings de un atributo si es una expresión Javascript debe ser en llaves, pero nunca juntas\***

**\*NOTA: ya que JSX es mas cercano a Jvascript, utiliza la convención de la nomenclatura camelCase en vez de nombres de atributos HTML. Por ejemplo class se vuelve className en JSX**

Otro dato interesante de **JSX**  es que previene los ataques de inyección, es seguro insertar datos ingresados por el usuario en **JSX**

Const tittle = response.potentiallyMaliciusInput;

Const element = <h1>{ tittle }</h1>

Por defecto, React DOM escapa / ignora cualquier dato insertado en JSX antes de ser renderizado, todo esto ( **JSX )**  es convertido en un string antes de ser renderizado. Esto ayuda a prevenir vulnerabilidades XSS ( cross-site-scripting ) , de este modo asegura que nunca se pueda insertar nada que no esté explícitamente escrito en tu aplicación

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elementos React**

Los elementos de React son inmutables. Una vez creas un elemento no puedes cambiar sus hijos o atributos, Un elemento es como un fotograma solitario en una película: este representa la interfaz de usuario en cierto punto en el tiempo.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Componentes y Propiedades**

**Completar leyendo** <https://es.reactjs.org/docs/react-component.html>

Los componentes permites separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada.

Conceptualmente los componentes son como las funciones de Javascript aceptan entradas arbitrarias ( osea parámetros ) llamadas “props” y devuellven a React elementos que describen lo que debe aparecer en la pantalla.

La forma más sencilla de definir un componente es escribir una función de Javascript:

* Function Welcome ( props ) {
  + Return <h1>Hello, { props.name }<h1>;
* }

Esta función es un componente de React válido ya que acepta un solo argumento (parámetro) de objeto “props” ( que proviene de propiedades ) con datos y devuelve un elemento de React. Se llaman estos componentes “funcionales” ya que literalmente son funciones Javascript.

También se puede utilizar una clase de ECMAScript6 (ES6) para definir un componente:

* Class Welcome extends React.Component {
  + Render() {
    - Return <h1>Hello, {this.props.name}</h1>
  + }
* }

**Renderización de componentes**

Los elementos también pueden representar componentes definidos por el usuario:

* Const elemento = <Welcome name=”Sara” />

Cuando React ve un elemento representando un componente definido (en este caso Welcome) pasa atributos JSX a este componente como un solo objeto. Este objeto es el que se entiende como props, el cual es recibido por el componente Welcome, entonces para definir, se puede ver que el elemento se incializará con un componente ya definido anteriormente llamado Welcome, al cuál le pasará un atributo que contiene un String el cuál lleva Sara, entonces desde el componente Welcome, se accederá a props que es el objeto que recibe un como argumento un componente y de este obtendrá ese atributo name.

\***NOTA: Comienza siempre los nombres de componentes con una letra mayúscula, ya que React toma los componentes que empiezan con letra minúscula como etiquetas HTML del DOM, en cambio si empieza por una mayúscula React tomará que es un componente que debe estar definido.\***

**Composición de Componentes**

Los componentes pueden referirse a otros componentes en su salida. En pocas palabras puedes usar componetes dentro de componentes.

Ejemplo ( tomando que el componente anterior Welcome)

Function App() {

Return(

<div>

<Welcome name=”Sara” />

<Welcome name=”Juan” />

<Welcome name=”David” />

</div>

);

}

Aquí se puede ver como el componente App, devolverá un div con el contenido que proviene de otros componentes.

\***Importante mirar el segmento de** <https://es.reactjs.org/docs/components-and-props.html>  **donde mencionan la importancia de dividir las cosas en componentes\***

**Recuerda, la idea de la creación de componentes es que sean cosas reutilizables así, si la web crece será más fácil reutilizar.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Estado y ciclo de vida**

**Importante averiguar bien este tema**

El estado de un componente es similar a los props pero es privado y está completamente controlado por el componente

**Convertir una función en una clase**

class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

render() {

return (

<div>

<h1>Hello, world!</h1>

<h2>It is {this.state.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

Puedes notar como se pasa el argumento del componente el cuál es props al constructor, los componentes de clase siempre deben invocar al constructor base con props.

class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

componentDidMount() {

this.timerID = setInterval(

() => this.tick(),

1000

);

}

componentWillUnmount() {

clearInterval(this.timerID);

}

tick() {

this.setState({

date: new Date()

});

}

render() {

return (

<div>

<h1>Hello, world!</h1>

<h2>It is {this.state.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

ReactDOM.render(

<Clock />,

document.getElementById('root')

);

¿Qué esta pasando con este código?

1. Cuando se pasa <Clock /> a ReactDOM.render(), React invoca al constructor del componente Clock. Ya que Clock necesita mostrar la hora actual, inicializa this.state con un objeto que incluye la hora actual. Luego actualizaremos este estado.
2. React invoca entonces al método render() del componente Clock. Así es como React sabe lo que se debe mostrar en pantalla. React entonces actualiza el DOM para que coincida con la salida del renderizado de Clock.
3. Cuando la salida de Clock se inserta en el DOM, React invoca al método de ciclo de vida componentDidMount(). Dentro de él, el componente Clock le pide al navegador que configure un temporizador para invocar al método tick() del componente una vez por segundo.
4. Cada segundo el navegador invoca al método tick(). Dentro de él, el componente Clock planifica una actualización de la interfaz de usuario al invocar a setState() con un objeto que contiene la hora actual. Gracias a la invocación a setState(), React sabe que el estado cambió e invoca de nuevo al método render() para saber qué debe estar en la pantalla. Esta vez, this.state.date en el método render() será diferente, por lo que el resultado del renderizado incluirá la hora actualizada. Conforme a eso React actualiza el DOM.
5. Si el componente Clock se elimina en algún momento del DOM, React invoca al método de ciclo de vida componentWillUnmount(), por lo que el temporizador se detiene.

**Usar el estado correctamente**

* **No modifiques el estado directamente:**
  + Por ejemplo:
    - This.state.comment = “Hello”;
  + En su lugar usa :
    - This.setState( {comment: ‘Hello’} );
  + El único lugar donde puedes asignar this.state es el constructor
* **Las actualizaciones del estado pueden ser asíncronas**
  + React puede agrupar varias invocaciones a setState() en una sola actualización para mejorar el rendimiento.
  + Debido a que this.props y this.state pueden actualizase de forma asincrónica, no debes confiar en sus valores para calcular el siguiente estado **IMPORTANTE.**
  + **Ejemplo**
    - // Incorrecto
    - this.setState({
    - counter: this.state.counter + this.props.increment,
    - });
  + **Correcto**
    - // Correcto
    - this.setState((state, props) => ({
    - counter: state.counter + props.increment
    - }));
  + Para arreglarlo, usa una segunda forma de setState() que acepta una función en lugar de un objeto. Esa función recibirá el estado previo como primer argumento, y las props en el momento en que se aplica la actualización como segundo argumento:
* **Las actualizaciones de estado se fusionan**
  + Cuando invocas a setState() React combina el objeto que proporcionaste con el estado actual, por ejemplo tu estado puede contener varias variables independientes.
  + Constructor(props){
    - Super(props);
    - This.state = {
      * Posts: [],
      * Comments: []
    - }
  + }
  + Luego estas se pueden actualizar de forma independiente con invoaciones separadas a setState();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<https://es.reactjs.org/docs/handling-events.html> quedé en esta parte.