# Notas – Curso de ReactJS

## Diferencia entre JavaScript Vanilla y React sin utilizar documentos .jsx

La diferencia uso de JavaScript vanilla y usar react sin documentos .jsx es que el código vanilla es muy imperativo, los statements que se generan utilizando el lenguaje de esta manera, se vuelve muy puntuales para lo que se debe hacer para una instancia muy puntual de la aplicación o página a desarrollar. El uso de react sin documentos .jsx, le puede permitir a JavaScript embeber de una manera más sencilla el HTML dentro de el lenguaje. Esto permite la facilidad de la maleabilidad de la información generada entre las etiquetas, sin embargo, también este código se vuelve no al mismo porcentaje de imperativo, pero, sí uno un poco más bajo.

## ¿Qué es JSX y de qué manera funciona?

Jsx es una extensión de ECMA script basada en XML la cual permite crear de una forma más declarativa y no imperativa los elementos que se renderizarán en la interfaz. De esta manera se pueden empaquetar de una manera más sencilla el HTML, CSS y JavaScript.

## ¿Qué son los transpiladores de JavaScript?

Son herramientas que permiten transformar código declarativo que escribimos con los documentos .jsx en código imperativo para ser ejecutado de esa manera. El proceso se resume en que se escribe inicialmente código de .jsx con su respectiva sintaxis y en el momento en el que se ejecuta en el navegador, este código, se transforma por medio del transpilador a código estructurado con la sintaxis de react sin estos documento para ser leído y renderizado.

## Notas

* Utilizando los documentos .jsx, los atributos de las etiquetas de los componentes de tienen que escribir en camelCase.
* Se puede incluir expresiones entre las etiquetas HTML utilizando { }. Hay que tener en cuenta que exclusivamente sólo se pueden escribir expresiones, cosas que devuelvan datos, no se pueden meter funciones, condiciones, ciclos o este tipo de cosas. Se pueden colocar variables, funciones que retornen valores etc…
* React evita la inyección de código, es decir, si se intenta renderizar una variable la cual contenga un script, algo como esto: const r = “<script>alert(‘´Hola’)</ script >” así { r }, esto devolverá el mismo contenido como un string.

## Crear aplicaciones de ReactJS

La manera oficial de react para crear aplicaciones es:

npx create-react-app {nombre\_app}

cd {nombre\_app}

npm start

Esta opción con el paso del tiempo, se ha quedado atrás en términos de rendimiento y velocidad. Otra opción, que es la mejor opción hoy en día para generar aplicaciones de react y es con Vite:

npm create vite@latest

Siguiendo el proceso de generación se puede crear una aplicación de react con typescript o javascript y utilizar swc o no en la elección que se escoja. Al generar la aplicación se crea una estructura de carpetas base para poder comenzar a generar el proyecto.

## Importando componentes dentro de otros

Los componentes generalmente son funciones que retornan contenido para ser renderizado por el navegador. Si se utilizará un componente generado por medio de una misma función desde el documento que se está utilizando, se debe tener en cuenta que el documento debe poseer una extensión .jsx, ya que esta nos permite indicar que vamos a utilizar la sintaxis de este mismo tipo de documentos. Los nombres poseen una particularidad y es que deben ser nombrados en PascalCase ya que react, en caso tal de utilizar por ejemplo un nombre en camelCase, este componente será renderizado como una etiqueta HTML nativa.

Si se va a importar un componente desde un archivo externo al que se está utilizando, se debe tener en cuenta que el nombre del documento .jsx deberá también tener esta nomenclatura en su nombre y dependiendo de la forma en la que se llame en el documento donde será invocado, deberá utilizarse una sintaxis u otra para su uso, así:  
  
Si la función que exporta el componente se llama “function App(){}”, primero debemos de agregarle el export para poderla utilizar en otros componentes: “export function App() {}” y ahora viene su implementación en los otros documentos:  
  
Si llamamos “ import App from ‘./App.jsx’ “ la función deberá de incluir “default” para ser exportada, así: “export default function App() {}”. Pero si llamamos al componente de la siguiente manera: “ import { App } from ‘./App.jsx’ “ sólo necesitaremos utilizar la sintaxis sólo con export que se mencionó anteriormente, así “export function App() {}”

## Pasándole props a un componente:

Este tema tiene una particularidad y por lo que se ha podido apreciar, no se trabaja puntualmente como aparece en el curso que solamente pasándole el prop en la invocación del componente es suficiente para que el proceso sea correcto. Según lo que se ha logrado entender, a parte de pasar el prop, hay que definir las propiedades que tendrán cada una de las props, así:

<TwitterFollowCard formatUserName={formatUserName} isFollowing userName="elonmusk" name="Elon Musk" />

TwitterFollowCard.propTypes = {

userName: PropTypes.string.isRequired,

name: PropTypes.string.isRequired,

isFollowing: PropTypes.bool.isRequired,

formatUserName: PropTypes.func.isRequired

};

Si se desea pasar un elemento como prop, el PropType sería “element”.

## Diferencia entre un componente y elemento:

La diferencia más característica que pueden tener ambas cosas es que un elemento es lo que renderiza react en el navegador y un componente se puede decir que actúa como una “fábrica” de elementos los cuales también son renderizados.