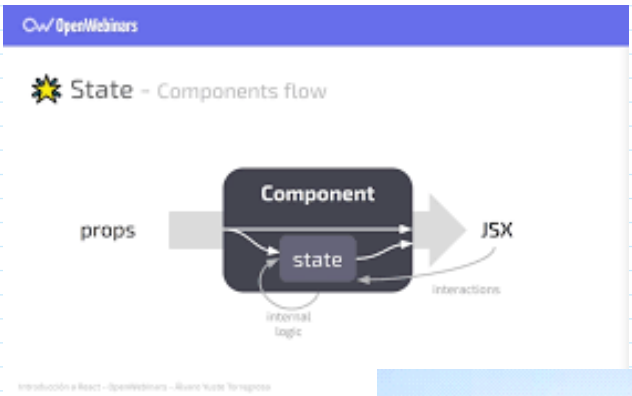
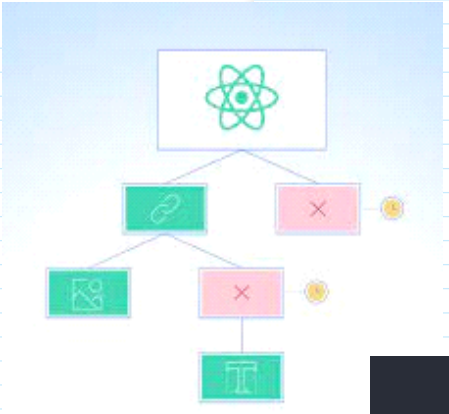
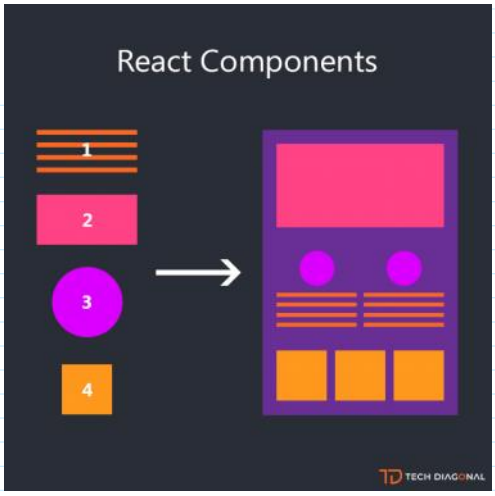
Trabajando con componentes

martes, 1 de marzo de 2022 15:21







¿Qué vamos a ver en esta clase pa?

Componentes -> Una pieza clave y fundamental para el desarrollo de aplicaciones del lado del front-end con React

- Qué son las props de un componente y por qué son tan importantes.
- Cómo podemos trabajar con las props de manera adecuada.
- Qué son los "Prop types" y las "Default props".
- Para qué sirve trabajar con los Children de un componente.
- Cómo trabajar con los estilos CSS de un componente.
- Y algunos secretillos más…

Las props de un componente

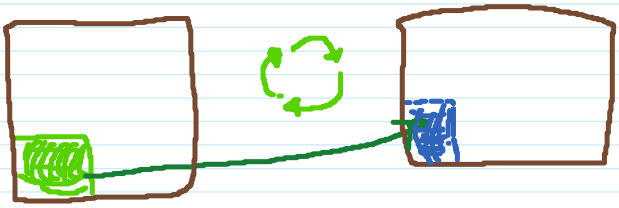
martes, 1 de marzo de 2022 15:55

Un componente lo podíamos utilizar para agilizar el proceso de nuestro trabajo

¿Cómo hacemos para que el contenido de un componente sea dinámico?

- Yo había leído que con los argumentos que le pasábamos a esa función propia de la estructura interna del componente
- Las props son datos internos del componente ✓
- Representan información que es enviada al momento en el que un componente es utilizado ✓
- Estas nos van a permitir que la información interna del componente **sea realmente variable** para que podamos tener estructuras HTML dinámicas y 100% reutilizables ✓

Un componente es un bloque de código que permite ser reutilizable



Pero ahora el tema es que.. No nos sirve para nada que ese componente que reutilizamos tenga siempre la misma información

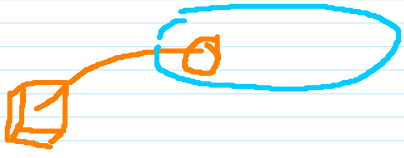
```
function Saludo () {
    return (
        <div>
            <h1>¡Hola mundo!</h1>
            <p>Qué lindo día para programar está haciendo hoy.</p>
        </div>
    );
```

Ahora, siempre que llamemos a este componente en cualquier lugar de la aplicación, el texto del saludo será el mismo. Ósea reutilizamos las cosas, pero podemos mejorar aún más el componente para no obtener algo estático todas las veces

props

Propiedades del componente!!! Es algo que nos ofrece React xd

Es un valor que se envía al componente en el mismo momento en que este se implementa o llama



Todos los componentes tienen la posibilidad de recibir los props por parámetro al momento

Todos los componentes tienen la posibilidad de recibir los props por parámetro al momento en que estos se están creando.

- La prop se trata de un objeto literal que contiene muchas propiedades que serán de utilidad para el componente (o que serán pasadas al componente) al momento de su uso.

{ a:,
b:
}

```
function Saludo (props) {
  return (
    <div>
        <h1> ¡Hola mundo! </h1>
        <p>Qué lindo dia para programar
           está haciendo hoy.</p>
    </div>
  );
}
```

Parámetro que pasamos al momento de crear el componente

No importa mucho el nombre, pero hay que llamarlo así para seguir con los estándares actuales de desarrollo

El componente puede recibir 2 propiedades, que ambas van a estar dentro de props

Luego lo que hay que hacer dentro del componente es imprimir la **propiedad** deseada en el lugar indicado

Lo que tenemos que hacer, mediante las llaves, es indicar el lugar donde queramos poner el valor de las propiedades, cuando las mismas se envíen

```
function Saludo (props) {
  return (
    <div>
        <h1> { props.titulo } </h1>
        <p> { props.texto } </p>
    </div>
  );
}
```

Ahora el componente saludo (o pieza de código) es

Reutilizable

Porque su contenido ya no es algo "fijo", sino que ahora va a cambiar de acuerdo a sus propiedades

¿En qué momento defino o le doy un valor a estas propiedades de título y texto?

Las propiedades de un componente reciben sus valores o se le asignan los mismos cuando el componente es invocado por la aplicación.

- Por ejemplo cuando importas el componente Saludo en el componente App. Es acá, en donde al utilizar el componente Saludo, podes pasarle sus respectivas propiedades o darle valor a las propiedades
  - Para hacerlo o lograrlo, alcanza con que al componente le configures en forma de atributo HTML: el nombre de la propiedad, y el valor que le quieras dar

```
<Saludo
   titulo="¡Hola mundo!"
   texto="Qué lindo día para programar está haciendo hoy"
/>
```

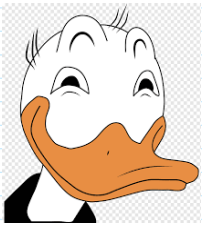
De esta manera, ya podemos hacer que un componente al momento de ser utilizado, reciba por parámetro distintos valores.

Y así poder enviar diversos saludos:

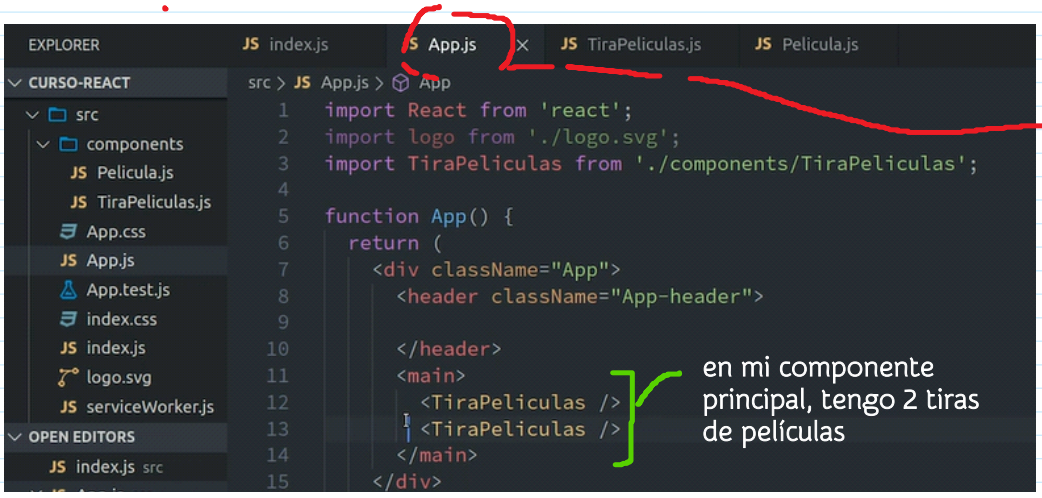
```
<Saludo
   titulo="¡Hola mundo!"
   texto="Qué lindo día para programar está haciendo hoy"
/>
<Saludo
   titulo= "Hello world!"
   texto=  "What a beatiful day to code"
/>
<Saludo
   titulo= " Bonjour le monde!"
   texto=  "Quelle belle journée pour coder"
/>
```

Podemos reutilizar este componente para diversas páginas y con diversos textos

- Ahora el componente sabrá que recibirá como parámetro, el valor de sus propiedades; y que los mostrará en el lugar donde le especificamos.



Practicando sobre las PROPS



```
EXPLORER
CURSO-REACT
  src
    components
      Pelicula.js
      TiraPeliculas.js
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    serviceWorker.js
OPEN EDITORS
  index.js src
  App.js src
  TiraPeliculas.js s...
  Pelicula.js src/co...
OUTLINE
```

```
src > JS App.js > App
1  import React from 'react';
2  import logo from './logo.svg';
3  import TiraPeliculas from './components/TiraPeliculas';
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9
10       </header>
11       <main>
12         <TiraPeliculas />
13         <TiraPeliculas />
14       </main>
15     </div>
16   );
17 }
18
19 export default App;
```

Es mi componente principal APP

en mi componente principal, tengo 2 tiras de películas

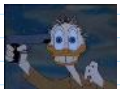
```
src > components > JS TiraPeliculas.js > TiraPeliculas
1  import React from 'react';
2  import Pelicula from './Pelicula';
3
4  function TiraPeliculas() {
5      return (
6          <div>
7
8              <Pelicula />
9              <Pelicula />
10             <Pelicula />
11             <Pelicula />
12             <Pelicula />
13         </div>
14     );
15 }
16
17 export default TiraPeliculas;
```

```
src > components > JS Pelicula.js > Pelicula
1  import React from 'react';
2
3  function Pelicula() {
4
5      let rating = 5.8;
6      let generos = ["Accion", "Drama"];
7
8      return (
9          <div>
10             <h2 className="sarasa">Titulo de la película!!</h
11             <p>Rating: {rating}</p>
12             <ul>
13                 { generos.map( genero => <li> {genero} </li>)
14             </ul>
15         </div>
16     );
17 }
18
19 export default Pelicula;
```

Esto debería modificarse según lo que estemos haciendo

Acá ponemos texto "genérico" y debemos reemplazarlo según lo amerite

Son datos que se van enviar al momento de generar el componente pa



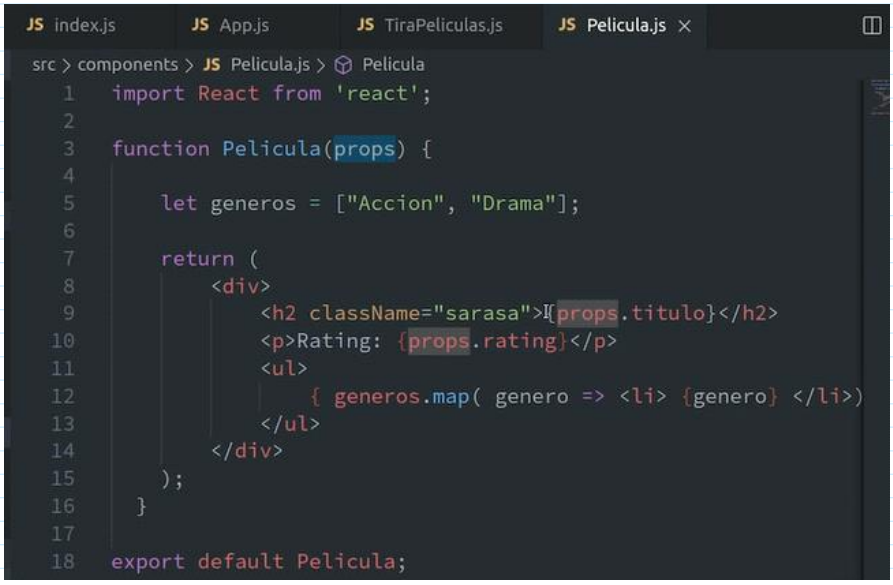
```
src > components > JS Pelicula.js > Pelicula
1  import React from 'react';
2
3  function Pelicula(props) {
4
5      let generos = ["Accion", "Drama"];
6
7      return (
```

Indica que vamos a tener

```
let generos = ["Accion", "Drama"];

return (
    <div>
        <h2 className="sarasa">Titulo de la película!!</h
        <p>Rating: {rating}</p>
        <ul>
            { generos.map( genero => <li> {genero} </li>)
        </ul>
    </div>
);
}

export default Pelicula;
```

Indica que vamos a tener algunas características particulares al momento de renderizar la película

Datos necesarios para luego ser renderizado

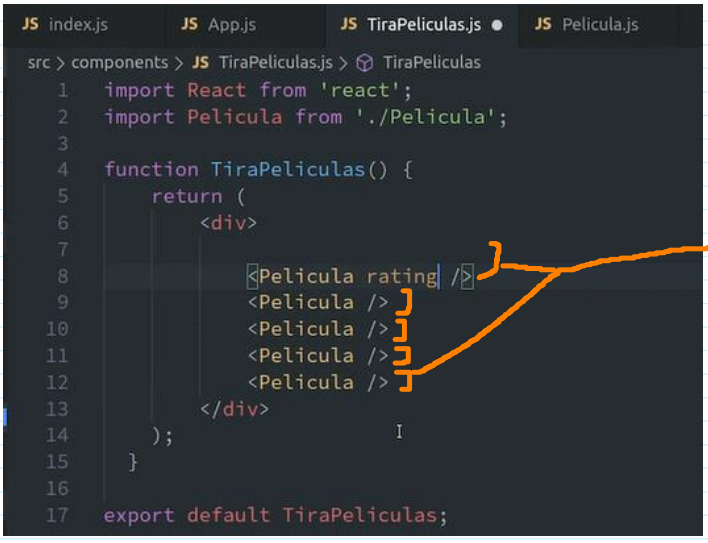


JS index.js | JS App.js | JS TiraPeliculas.js | JS Pelicula.js

src > components > JS Pelicula.js > Pelicula

```
import React from 'react';

function Pelicula(props) {

    let generos = ["Accion", "Drama"];

    return (
        <div>
            <h2 className="sarasa">{props.titulo}</h2>
            <p>Rating: {props.rating}</p>
            <ul>
                { generos.map( genero => <li> {genero} </li>)
            </ul>
        </div>
    );
}

export default Pelicula;
```

¿Cómo hacemos para ejecutarlo o invocarlo?



JS index.js | JS App.js | JS TiraPeliculas.js ● | JS Pelicula.js

src > components > JS TiraPeliculas.js > TiraPeliculas

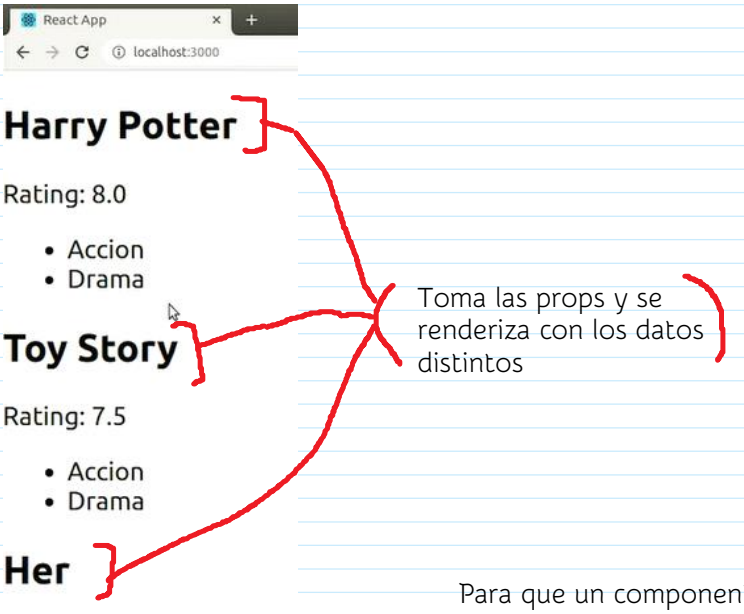
```
import React from 'react';
import Pelicula from './Pelicula';

function TiraPeliculas() {
    return (
        <div>

            <Pelicula rating />
            <Pelicula />
            <Pelicula />
            <Pelicula />
        </div>
    );
}

export default TiraPeliculas;
```

Acá están las invocaciones y acá es donde vamos a colocar las props

```
JS index.js   JS App.js   JS TiraPeliculas.js ●   JS Pelicula.js
src > components > JS TiraPeliculas.js > TiraPeliculas
1   import React from 'react';
2   import Pelicula from './Pelicula';
3
4   function TiraPeliculas() {
5       return (
6           <div>
7
8               <Pelicula rating="8.0" titulo="Harry Potter" />
9               <Pelicula rating="7.5" titulo="Toy Story"/>
10              <Pelicula rating="6.0" titulo="Her"/>
11              <Pelicula rating="9.0" titul/>
12              <Pelicul abc titulo
13          </div>
14      );
15  }
16
17  export default TiraPeliculas;
```

Le damos props a c/película

Vamos a tener la misma estructura para los componentes sólo que con un contenido distinto

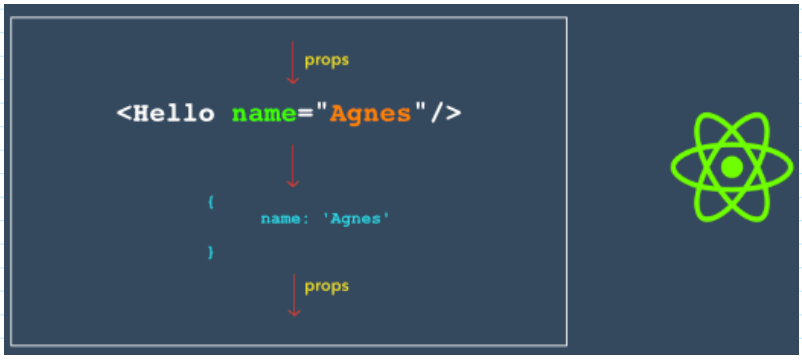


Toma las props y se renderiza con los datos distintos

Para que un componente sea 100% reutilizable es sólo cuestión de utilizar las **props**

Mini resumen

Las props son entradas de un componente de React. Representan la información pasada desde un componente padre a un componente hijo. Pueden recibir propiedades como parámetros para poder insertar valores y eventos en el HTML



Esta función es un componente de React válido porque acepta un solo argumento de objeto "props" (que proviene de propiedades) con datos y devuelve un elemento de React.

Viene de **propiedades**

```
function Bienvenido(props) {
 return (
   <div>
       <h1>
         Hola, { props.nombre }
       </h1>
   </div>
 );
}
```

Argumento de objeto "props" que contiene datos y que es aceptado como un componente valido de React.

Devuelve un elemento de React

Este componente no retornará texto fijo, sino que el mismo retornará contenido de acuerdo al valor de las propiedades

Una forma más sencilla de definir una **props** es que son funciones de JavaScript. Y al decir esto, sumamos el concepto de componente funcional

Implementación de PROPS

```
<Bienvenido
     nombre =  "Victor Luis"
     />
```

Las propiedades del componente reciben su respectivo valor, cuando el componente es invocado por la aplicación

Acá cuando llamamos al componente es cuando podemos definir el valor que va a llevar esa props.

Pregunta, las props son solo de lectura, no? Osea no podes crear un nuevo elemento en base las props?

Cuando queremos definir las props de un componente, lo tenemos que hacer como si fuese un atributo de HTML, teniendo en cuenta que si viésemos el objeto props, en este caso, quedaría algo así: Props = { título: "Harry Potter" }

1 ¿Qué son las props?

- Son las propiedades de un componente.
- Es información que le envía un componente hijo al padre.
- Es información que le envía un componente padre a un hijo.

2 Tengo que enviar mucha información a un componente, por lo que pienso hacerlo en varias props. ¿Es correcto lo que pienso?

- Sí, para ser performantes, si hay mucha información, es mejor enviar varias props.
- No, la información se envía una sola vez a un componente a través de props.

3 ¿Cuándo se define la información que va a ser enviada por prop?

- Cuando se crea el componente.

4 Props es...

- un array.
- un objeto literal.

5 ¿Cuál de las siguientes sentencias es incorrecta?

- <Libro props={ {titulo:"Harry Potter"} }/>

6 Supongamos que tenemos un componente que se llama **Saludo** y queremos pasarle dos props. Las props que queremos pasarle serían las siguientes:

titulo = "Hola mundo"

subtitulo = "¡Nunca paremos de aprender!"

Te invitamos a que abras tu editor de texto y que pruebes de crear el componente **Saludo** pasándole esas dos props. Luego, te vamos a pedir que pegues, en el campo debajo, cómo hiciste para definir las props de este componente.

Te dejamos la estructura del componente **Saludo**:

```
function Saludo(props) {
 return (
   <div>
       <h1>{props.titulo}</h1>
       <h1>{props.subtitulo}</h1>
   </div>
 );
}

export default Saludo;
```

69/1500 caracteres

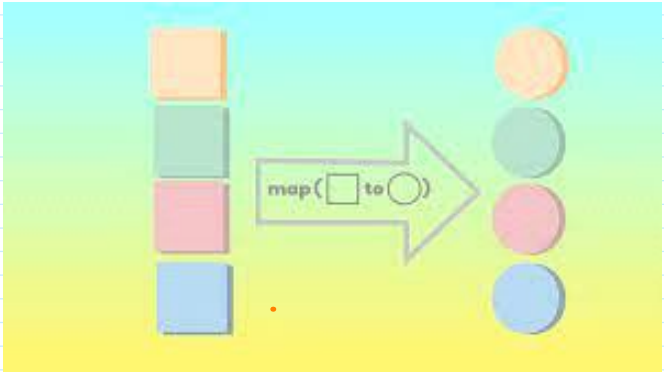
```
<Saludo titulo="Hola mundo" subtitulo="¡Nunca paremos de aprender!"/>
```

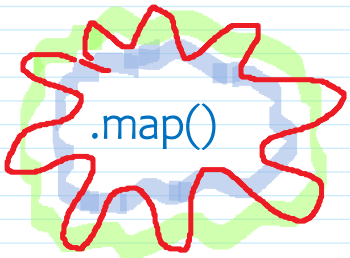
Key props y .map()

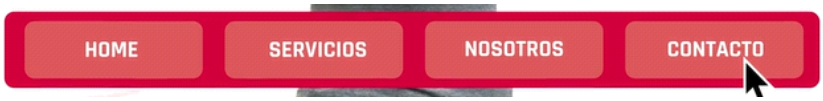
miércoles, 2 de marzo de 2022 19:08

¿q verga son las key props?

- Imagina que dentro de tu componente, a través de las props, recibís un array de objetos literales que representan la base datos que quieres mostrar en la vista
  - Ahora que tenemos esa información dentro del componente, nuestro siguiente objetivo será que por cada objeto literal del array, se renderice una etiqueta <li> del HTML
    - Algo importante es que dentro de un componente de React, es imposible usar iteradores. En su lugar habrá que utilizar métodos que iteren, asincrónicamente, la información



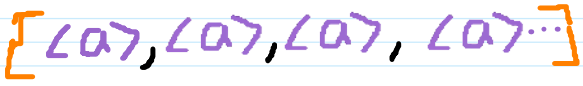




- Los elementos que se repiten constantemente son los enlaces

Ahora si queremos crear un componente NAVBAR, que pueda ser reutilizado en varios contextos ¿Cuántos enlaces crearías? ¿Cómo mandarías/enviarías al componente, el contenido con todos los enlaces?

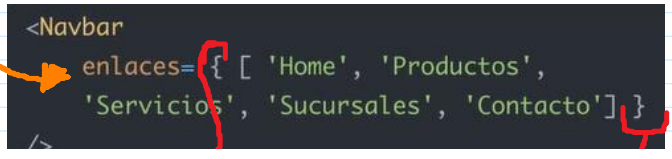
- Vamos a estar parados en el escenarios en donde no sabes con exactitud, que cantidad de elementos que va a estar renderizando tu componente. Es acá donde se hace necesario trabajar con arrays y sus métodos.
- De esta forma ya no te va a interesar que al componente le pasemos una cantidad exacta de enlaces, ya que vos vas a saber que las mismas van a ser pasadas como un array a través de las props



- Y dentro del componente, tu trabajo será, iterar sobre ese array e imprimir la cantidad de enlaces que correspondan (tanto como se hayan enviado)

¿Cómo se van a enviar los enlaces desde los atributos del componente?

Recibe por atributo, un array con los enlaces que se desean renderizar



con los enlaces que se desean renderizar

```
enlaces={ [ 'Home', 'Productos',
'Servicios', 'Sucursales', 'Contacto'] }
/>
```

Permiten escribir tipos de datos JS, que no sean simplemente cadenas de texto

Veamos el componente, en sí, que tiene que imprimir esos enlaces

Hay que iterar sobre el array recibido en el parámetro "props" para poder imprimir estos enlaces

El mejor tipo de ciclo que itera sobre un array e imprimir su contenido dentro de React es, map.

- Este método itera sobre c/u de los elementos del array y *retorna un resultado por c/u* de los elementos. Y es acá, en el retorno, donde el elemento podrá renderizarse correctamente

```
function Navbar (props) {
    return (
        <nav>
            <ul>
                {
                    props.enlaces.map(unEnlace => <li> {unEnlace} </li>)
                }
            </ul>
        </nav>
    )
}
```

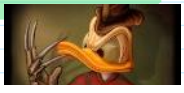
Propiedad que es un array

Es un array, y es lo que recibe el componente

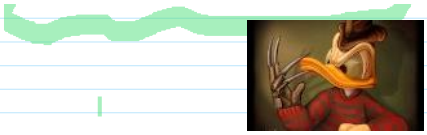
```
function Navbar (props) {
    return (
        <nav>
            <ul>
                {
                    props.enlaces.map(unEnlace => <li> {unEnlace} </li>)
                }
            </ul>
        </nav>
    )
}
```

Se está retornando una estructura JSX, que contiene al elemento

Y entre las etiquetas <li> se está imprimiendo a cada enlace, haciendo uso de las llaves para poder mostrar el texto correspondiente



Y entre las etiquetas <li> se está imprimiendo a c/u de los enlaces, haciendo uso de las "llaves" para así poder mostrar el texto correspondiente



el texto correspondiente

OSEA, el método map, nos va a retornar por cada uno de los elementos del array, un bloque de JSX

- HOME
- PRODUCTOS
- SERVICIOS
- SUCURSALES
- CONTACTO

```
Warning: Each child in a list should have a
unique "key" prop.

Check the render method of `Navbar`. See
https://fb.me/react-warning-keys for more
information.
    in li (at Navbar.js:8)
    in Navbar (at App.js:23)
    in div (at App.js:16)
    in App (at src/index.js:8)
```

Cuando un componente va a renderizar varios elementos del mismo tipo (como en el ejemplo, tenemos varios <li>, pero no sabemos cuántos), van a necesitar de una key con un valor único. Porque esto le va a ayudar a identificar a React cuáles son los items que cambiaron/se agregaron/se eliminaron

★ Las keys deben ser dadas a los elementos, dentro del mapeo del array. Para darle a los elementos una identidad única y estable.

¿Cómo saco una key única y estable?

- La única forma es utilizando un string que identifique, unívocamente, al elemento entre sus hermanos

Vas a tener que utilizar el atributo **key**, el cual vas a tener que utilizarlo de esta manera:

-
```
function Navbar (props) {
    return (
        <nav>
            <ul>
                {
                    props.enlaces.map((unEnlace, i) => <li key={unEnlace + i}> {unEnlace} </li>)
                }
            </ul>
        </nav>
    )
}
```

```
        </nav>
    )
}
```

Le estas dando un valor único

Es el texto correspondiente a cada enlace + el índice en el cual se encuentra el mismo dentro del array

Las keys quedan:

Home0
Productos1
Servicios2
Sucursales3
Contacto4

Ahora cada elemento <li> que se está renderizando, cuenta con un identificador único

Que lo hace diferente de sus demás elementos hermanos

Sólo con eso de la KEY, React va a trabajar como se debe

Aclaraciones

- Este atributo <key>, nunca se imprime como atributo HTML. Ósea, no lo vas a ver en el navegador, en el HTML, devtools. Es sólo de uso exclusivo para React

Ejemplos en código

index.js | App.js | TiraPeliculas.js | Pelicula.js

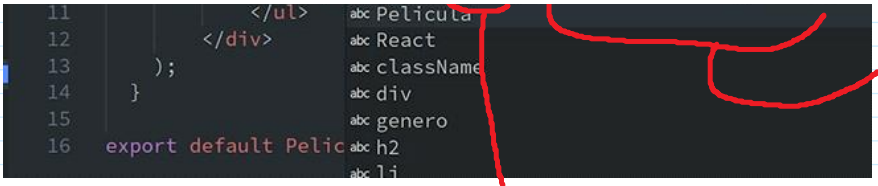
src > components > Pelicula.js > Pelicula

```
import React from 'react';

function Pelicula(props) {

    return (
        <div>
            <h2 className="sarasa">{props.titulo}</h2>
            <p>Rating: {props.rating}</p>
            <ul>
                { props.generos.map( genero => <li> {genero}
            </ul>
        </div>
    );
}

export default Pelic
```

Autocomplete: Pelicula, React, className, div, genero, h2, li

Dentro de props, ya tengo un listado con los géneros

Por cada uno, disparamos un <li> con el nombre del género



Por cada uno, disparamos un <li> con el nombre del género

Cada uno de los géneros va a ser procesado a través map. Y map nos va a dar un resultado para c/u de los géneros

Vamos a hacer que lleguén cosas al props

Cada uno de estos géneros va a ser impreso de una variable de JS

Según la película, los géneros van a ser distintos

Harry Potter

Rating: 8.0

- Fantasia
- Aventuras

Toy Story

Rating: 7.5

- Animación
- Familiar

Her

Cada una de las películas renderiza de acuerdo a sus géneros

Las props nos permiten dar información personalizada para cada uno de los componentes

Listas = array de elementos

Dentro del resultado del map, decimos qué output esperamos por cada uno de los items de la lista

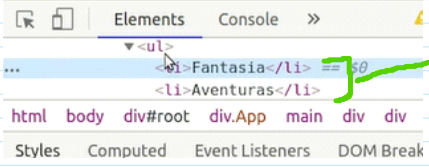
```
<ul>
    { props.generos.map( genero => <li> {genero} </li>
</ul>
```

Elementos que se repiten en nuestro output

Trabajamos con el map para hacer un **output** para cada uno de los elementos

```
JS index.js   JS App.js   JS TiraPeliculas.js   JS Pelicula.js ×
src > components > JS Pelicula.js > Pelicula
1   : from 'react';
2
3   .icula(props) {
4
5   .
6   />
7    <h2 classNam (parameter) props: any }</h2>
8    <p>Rating: {props.rating}</p>
9    <ul>
10       { props.generos.map( (genero, i) => <li key={genero + i}
11   </ul>
12  iv>
13
14
15
16  ilt Pelicula;
```

iterador



No nos aparece el atributo **key** porque es algo propio de React

React use key, que es único para cada <li> del componente, para saber qué renderizar y hacerlo de forma optima

- Cada vez que tengamos un array, recibido por prop o en una variable, vamos a usar el map para hacer un output para c/u de los elementos
- De esta manera podes hacer que un componente itere sobre un array y renderice, en el navegador, la información esperada

Mini resumen

```
const usuarios = ["Maru","Uriel","Daniel"];
<MiLista
     items = { usuarios }
     />
```

Así es como se define un valor para props a través de un array

Al componente le estamos pasando usuarios en su atributo ítems

Son estas llaves las que nos permiten escribir tipos de datos que no sean simplemente "cadenas de texto"

Dentro del componente:

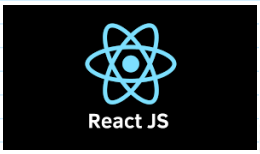
-

```
function MiLista(props) {
 return (
   <div>
     <ol>
       {props.items.map(item => <li> {item} </li>)}
     </ol>
   </div>
 );
}
```

Vamos a tener que recibir sobre el array recibido para poder imprimir los usuarios

Es un método que conviene utilizar en React. Y es para iterar sobre un array y devolver su contenido

Las **keys** ayudan a React a identificar qué elementos se han modificado, agregado o eliminado.



Un poco más sobre las keys:

- Por medio de estas React identifica si se trata de un mismo elemento/componente o no
- Es necesario implementar las keys cuando devolvemos un array de elementos que son iguales
- La key debe ser única entre elementos hermanos
- La key no está visible en el HTML, para que sea visible, en lugar de key sería id

```
function MiLista(props) {
 return (
  <div>
   <ol>
    {props.items.map((item, i)=><li key={i+item}> {item}</li>)}
   </ol>
  </div>
 );
}
```

Los componentes que renderizan varios elementos del mismo tipo necesitan una key **con valor único.** Ya que ayuda a react a identificar las acciones que se realizaron sobre elementos puntuales

La key se le da al elemento dentro del mapeo para generar una key única y estable

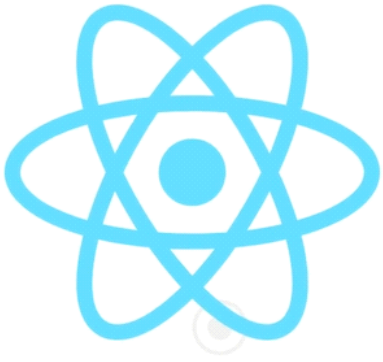
Las PropTypes y las DefaultProps

viernes, 4 de marzo de 2022 14:30

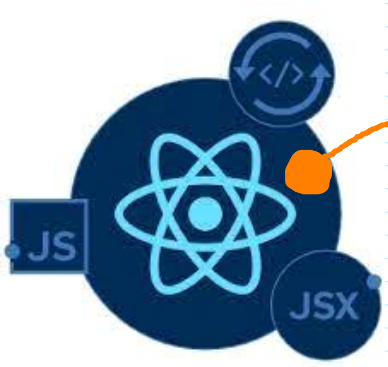
¿Qué pasaría si dentro de un componente, en el lugar que esperamos recibir un array para iterarlo, recibieras un string? ¿El método .map() seguiría funcionando? ¿El componente renderizaría correctamente?

Lo solucionamos con las famosas **PropTypes**, que con ellas determinamos específicamente el tipo de dato de una props

También tenemos **DefaultProps**, que nos van a permitir definir valores por defecto para determinadas props



Cada vez que tu aplicación de React va creciendo, es bueno tener más control sobre las propiedades que los componentes reciben. De esa forma aseguramos la calidad



Dentro del ecosistema de React tenemos las **PropTypes**

Y es una manera en la que podés definir el tipo de dato que se espera recibir para una determinada prop de un componente

```
function(type prop)
{
}
```

O por ejemplo, un componente que recibe un título:

```
function(type titulo)
{
}
```

Acá sería un string

Obviamente esto es props y dentro de este, esta titulo

Hay que instalar un paquete NPM para poder tener acceso a **propTypes** :"v

Instalación e implementación

- Para hacerlo, antes tenes que estar en el directorio raiz de tu proyecto de React
  - `npm install prop-types --save`
- Para usarlo o implementarlo hay que importarlo en el componente donde queramos utilizar este paquete

propTypes no es necesario que este implementado en todos los componentes	**propTypes** es más bien como una ayuda a nivel de escritura de código, y que nos permite estar al tanto de posibles errores en el uso de nuestros componentes

```
1  import propTypes from 'prop-types';
2
3
```

Paquete importado en el componente. Y alcanza definir que para ese componente queres hacer uso de las propTypes

Definiendo el tipo de dato una prop

- Antes de la exportación del componente hay que asignarle al componente una propiedad llamada propTypes
  - propTypes

```
MiComponente.propTypes = {...}
```

Como valor siempre es un objeto literal

```
MiComponente.propTypes = {...}
```

Y acá vamos a tener que hacer referencia al nombre de la prop a la cual le quieres especificar el tipo de dato. Y a la prop le tenes que asignar, por medio del paquete propTypes, el tipo de dato especifico

```
1  Navbar.propTypes = {
2    enlaces: propTypes.array
3  }
```

★ Observa como la propiedad asignada al nombre del componente, no va en MAYUSCULAS. Mientras que la referencia al paquete importado, sí.

Navbar

```
import React from 'react';
import propTypes from 'prop-types';

function Navbar (props) {
  return (
    <nav>
      <ul>
        {
          props.enlaces.map((unEnlace, i) => {
            return (
              <li key={unEnlace + i}><a href={unEnlace.url} {unEnlace.texto} </a></li>
            )
          })
        }
      </ul>
    </nav>
  )
}

Navbar.propTypes = {
  enlaces: PropTypes.array
}

export default Navbar;
```

Aquí le faltó cerra la etiqueta <a>

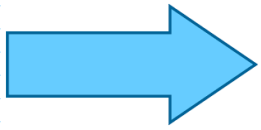
El mismo Darío dijó que si no seguimos las convenciones de sintaxis, las cosas no pueden llegar a funcionar

propTypes te da un cumulo de opciones que podes utilizar, ya que no sólo te da la opción de definir el tipo de dato que se espera recibir en una prop, sino que podes definir que dicha prop sea:

- Obligatoria
- Que espera recibir un array de un cierto tipo de dato
- O que esa prop pueda recibir uno de varios tipos de datos posibles

Cúmulo: Gran acumulación o reunión de cosas materiales o inmateriales, especialmente cuando están superpuestas unas sobre otras en el espacio o en el tiempo.

Definiendo los valores por defecto que tendra una prop de un componente



Esta función viene por defecto en cualquier componente de React

Sólo hay que definir una propiedad para el componente que se llame **defaultProps**.

```
MiComponente.defaultProps = {...}
```

Acá haces referencia al nombre de la propiedad y le asignas el valor por defecto que querés que la propiedad tenga


```
1  Saludo.defaultProps = {
2    titulo: '¡Hola mundo!',
3    mensaje: 'Saludos desde el planeta React',
4  }
5
```

- Esto es para definir valores por defecto para cada propiedad que vos quieras, en caso de que las mismas no reciban valores al momento de que el componente es utilizado

Es una forma elegante de evitar que la renderización de los componentes fallen, por la omisión de valor en las props

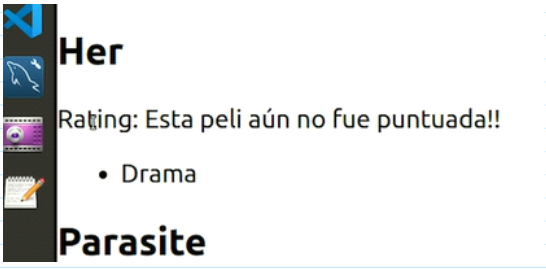
Ejemplo práctico

```
JS index.js  JS App.js  JS TiraPeliculas.js  JS Pelicula.js ●
src > components > JS Pelicula.js > Pelicula
1  import React from 'react';
2
3  function Pelicula(props) {
4
5      return (
6          <div>
7              <h2 className="sarasa">{props.titulo}</h2>
8              <p>Rating: {props.rating}</p>
9              <ul>
10                 { props.generos.map( (genero, i) => <li key={
11             </ul>
12         </div>
13     );
14   }
15
16 Pelicula.defaultProps = {
17     rating: "Esta peli aún no fue puntuada"
18 }
```

En caso de que no me den rating de una película

Para probarlo tenemos que ir a la invocación de los componentes y borrar el rating

```
JS index.js  JS App.js  JS TiraPeliculas.js ×  JS Pelicula.js
src > components > JS TiraPeliculas.js > TiraPeliculas
1  import React from 'react';
2  import Pelicula from './Pelicula';
3
4  function TiraPeliculas() {
5      return (
6          <div>
7
8              <Pelicula generos={["Fantasia", "Aventuras"]} rat
9              <Pelicula generos={["Animación", "Familiar"]} rat
10             <Pelicula generos={["Drama"]} rating="6.0" titulo
11             <Pelicula generos={["Drama"]} rating="9.0" titulo
12             <Pelicula generos={["Acción", "Aventura"]} rating
13         </div>
14     );
15   }
16
17 export default TiraPeliculas;
```



```
JS index.js  JS App.js  JS TiraPeliculas.js  JS Pelicula.js ●
src > components > JS Pelicula.js > Pelicula > constructor
 4
 5      return (
 6          <div>
 7              <h2 className="sarasa">{props.titulo}</h2>
 8              <p>Rating: {props.rating}</p>
 9
10              <ul>
11                  { props.generos.map( (genero, i) => <li key={
12              </ul>
13          </div>
14      );
15  }
16
17  Pelicula.defaultProps = {
18      rating: "Esta peli aún no fue puntuada!!",
19      generos: null
20  }
21
22  export default Pelicula;
```

Hay que hacer un condicional, qué si tenemos géneros se haga el

Y si no los tiene que no se imprima nada

```
JS index.js  JS App.js  JS TiraPeliculas.js  JS Pelicula.js ×
src > components > JS Pelicula.js > Pelicula > constructor
 1  import React from 'react';
 2
 3  function Pelicula(props) {
 4      let listadoDeGeneros;
 5      if (props.generos != null) {
 6          let listadoDeGeneros =
 7          <ul>
 8              { props.generos.map( (genero, i) => <li key={gene
 9          </ul>
10      } else {
11          let listadoDeGeneros = "";
12      }
13
14      return (
15          <div>
16              <h2 className="sarasa">{props.titulo}</h2>
17              <p>Rating: {props.rating}</p>
18              { listadoDeGeneros }
19          </div>
```

05:54

Muy interesante la asignación

Es como un condicional una vez conocido el valor que fue pasado o si estamos en presencia del valor por defecto.

Si queremos presumir, podemos convertir a listadoDeGeneros en un componente con toda esta logica, de esa forma estamos dividiendo/modularizando los componentes para hacerlos más reutilizables

Un poquito de propTypes

```
src > components > Pelicula.js > Pelicula > constructor
1  import React from 'react';
2  import PropTypes from 'prop-types';
3
4  function Pelicula(props) {
5
6      let listadoDeGeneros;
7
8      if (props.generos != null) {
9          listadoDeGeneros =
10         <ul>
11             { props.generos.map( (genero, i) => <li key={gene
12         </ul>
13     } else {
14         listadoDeGeneros = "";
15     }
16
17     return (
18         <div>
19             <h2 classNam="sarasa">{props.titulo}</h2>
```

```
src > components > Pelicula.js > Pelicula
18             <div>
19                 <h2 className="sarasa">{props.titulo}</h2>
20                 <p>Rating: {props.rating}</p>
21                 { listadoDeGeneros }
22             </div>
23         );
24     }
25
26 Pelicula.defaultProps = {
27     rating: "Esta peli aún no fue puntuada!!",
28     generos: null
29 }
30
31 Pelicula.propTypes = {
32     rating: PropTypes.number,
33     titulo:
34 }
```

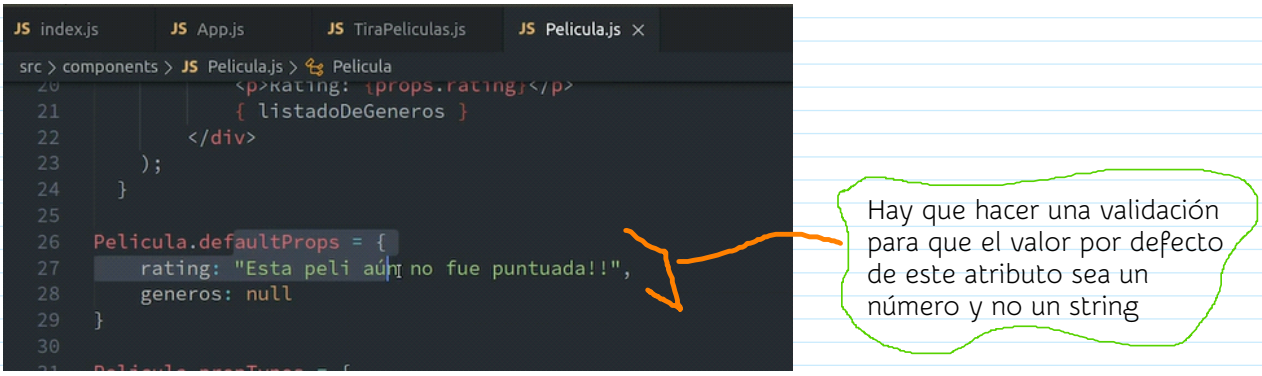
Indicamos el tipo de dato para c/u de estos atributos

Es para hacer un programa consistente (duro, resistente, compacto) y que a los demás les baje las probabilidades de equivocarse

```
src > components > TiraPeliculas.js > TiraPeliculas
1  import React from 'react';
2  import Pelicula from './Pelicula';
3
4  function TiraPeliculas() {
5      return (
6          <div>
7
8              <Pelicula generos={["Fantasia", "Aventuras"]} rat
9              <Pelicula generos={["Animación", "Familiar"]} rat
10             <Pelicula generos={["Drama"]} titulo="Her"/>
11             <Pelicula rating="asjidoasj" titulo="Parasite"/>
12             <Pelicula generos={["Accion", "Aventura"]} rating
13         </div>
14     );
15 }
16
17 export default TiraPeliculas;
```

```
<Pelicula generos={["Fantasia", "Aventuras"]} rating={8.0} t
<Pelicula generos={["Animación", "Familiar"]} rating={7.5} t
<Pelicula rating={} generos={["Drama"]} titulo="Her"/>
<Pelicula rating={9.0} titulo="Parasite"/>
<Pelicula generos={["Acción", "Aventura"]} rating={4.1} titu
iv>
```

DE esta forma no explota



Nuestro código debe ser prolijo, para que nuestros compañeros estén alineados con lo que el código espera

★ Así aseguramos o le damos una capa de seguridad/prolijidad cuando trabajamos con React

- Y es así como podes implementar y definir el tipo de dato, que una prop de un componente, espera recibir. Y también los valores por defecto, si es que dicha prop no recibe ningún tipo de valor al momento de utilizar el componente

Mini resumen

- A medida que nuestra aplicación crece, se vuelve necesario capturar los errores mediante la verificación de tipos. React tiene fuertes habilidades para la verificación de tipos de datos en las props de un componente. Es simplemente asignar la propiedad **propType** a nuestro componente

Las PropTypes pertenecen al conjunto de paquetes que gestiona **npm**, por lo que debemos utilizarlo para instalarlas.

```
npm install --save prop-types
```

```
import PropTypes from 'prop-types'; // ES6

var PropTypes = require('prop-types'); // ES5 with npm
```

Raro xd

```
import PropTypes from 'propp-types';;

Function Saludar(props) {
  render() {
    return (
      <h1>Hola, {props.nombre}</h1>
    );
  }
}
Saludar.propTypes = {
  nombre : PropTypes.string
};
export default Saludar;
```

PropTypes exporta un rango de validadores que pueden ser usados para estar seguros de que la información que es recibida sea valida

DefaultProps

Es una propiedad en el mismo componente, para establecer las props predeterminadas que recibirá el mismo

Se utiliza para props no definidos, pero no para props nulos

```
function SeteoBoton(props)  {
  // ...
}

SeteoBoton.defaultProps = {
  color: 'blue'
};
```

Se hace referencia al nombre de la propiedad y se le asigna el valor por defecto que queremos que la propiedad tenga

1 ¿Es verdad que las prop types pueden setear valores por defecto en caso de que un componente no tenga valor?

- Sí, es verdad.
- No, es mentira.

2 ¿Qué tipo de dato aceptan las prop types y las default props?

- Un array.
- Un objeto literal.

3 ¿Cuál de las siguientes líneas de código es la correcta?

```
Componente.propTypes = {
  nombre: propTypes.string
}
```

Children

viernes, 4 de marzo de 2022 16:58

Va a haber casos en donde queramos componentes que tengan un uso bastante dinámico que no nos va a alcanzar con el concepto de **prop** que habíamos visto. Es por eso que nos remontaremos en aprender un poquito sobre los famosos **children**

A través de estos vamos a ser capaces de enviar, a modo de prop, cualquier estructura HTML

```
<Saludo
 titulo="¡Hola mundo!"
 mensaje="Que lindo día para programar"
/>

<Saludo />
```

valores

atributos

Asi es como usamos un componente

```
<Saludo
  <h2>¿Funcionará?</h2>
</Saludo>
```

¿Qué hace esto?

- En el navegador no se muestra nada :V

- Para React, todo lo que sucede al momento de usar un componente, debe ser especificado al momento de su creación (del componente)
  - Ósea que si sabes que el componente va a tener las props y contenido entre sus etiquetas, esto hay que especificarlo

CHILDREN: Son aquellos elementos que son pasados entre las etiquetas del componente, y no a modo de props sino como hijos del componente. Y hay que hacer uso de las props para preparar al componente para recibir hijos

Todo componente, dentro de este objeto literal **props**, tiene una propiedad llamada **children.** El cual me traera consigo todos los elementos que se hayan dispuesto entre las etiquetas del componente, al momento de que el mismo haya sido utilizado

```
function Saludo (props) {
  return (
    <div>
      <h1>{ props.titulo }</h1>
      <p>{ props.mensaje }</p>
      { props.children }
    </div>
  );
}
```

Función que define al componente

Le dice al componente "que tiene por asegurado que va a recibir hijos, y que los imprima en ese lugar"

★ Y sea un solo children o muchos, los mismos se van a imprimir en ese mismo orden en el que fueron enviados, ya que todos se mandan en la misma prop

- Tener acceso a esta prop del componente, lo vuelve algo muy poderoso, ya que no solo vas a tener un conjunto de props dinámicas, sino que vas a tener la posibilidad de que el componente reciba la cantidad de hijos que sean necesarios

Ejercicio práctico

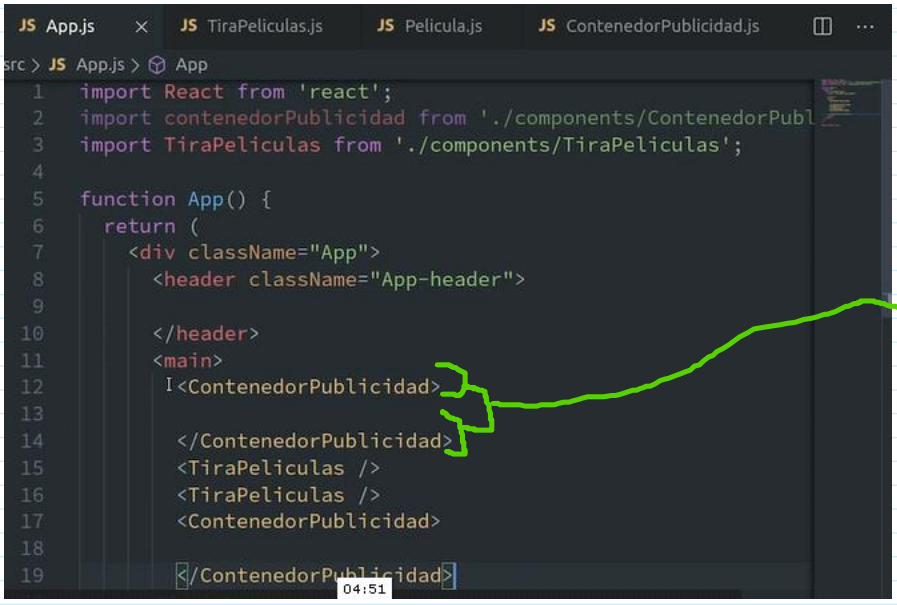
El uso de la propiedad children aparece, como necesario, cuando no sabemos qué es lo que va a tener nuestro componente

Si no sabemos qué va a tener nuestro componente, o mi código tiene muchos condicionales o cuestiones que compliquen mucho la lectura: nos conviene las children

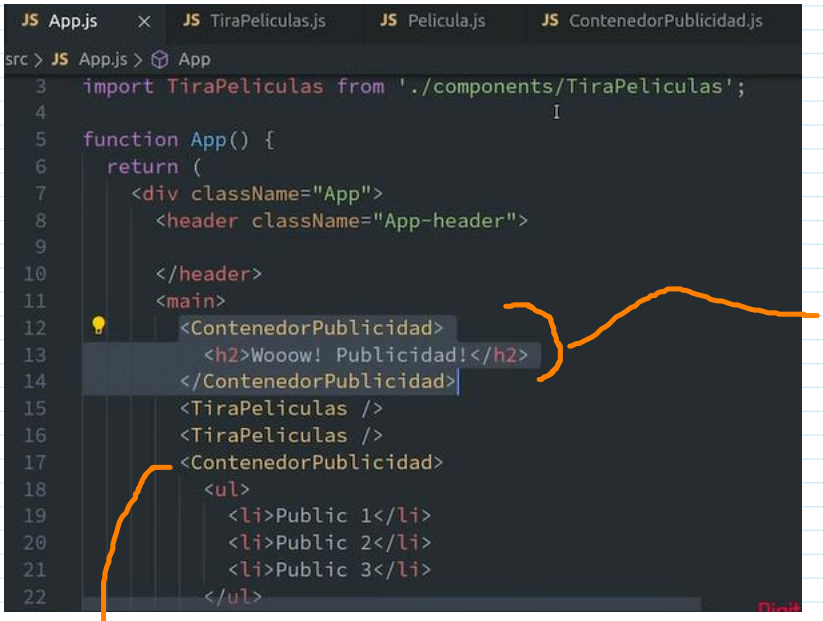
Lo que va a tener internamente el componente

Lo que yo no sé en este componente, es que según la publicidad, acá puede haber un GIF, VIDEO, IMAGEN o texto

Acá es buena idea utilizar la **prop.children**, ya que yo no se qué es lo que va a estar acá dentro xd. Mejor que andar pensando billones de condicionales xd

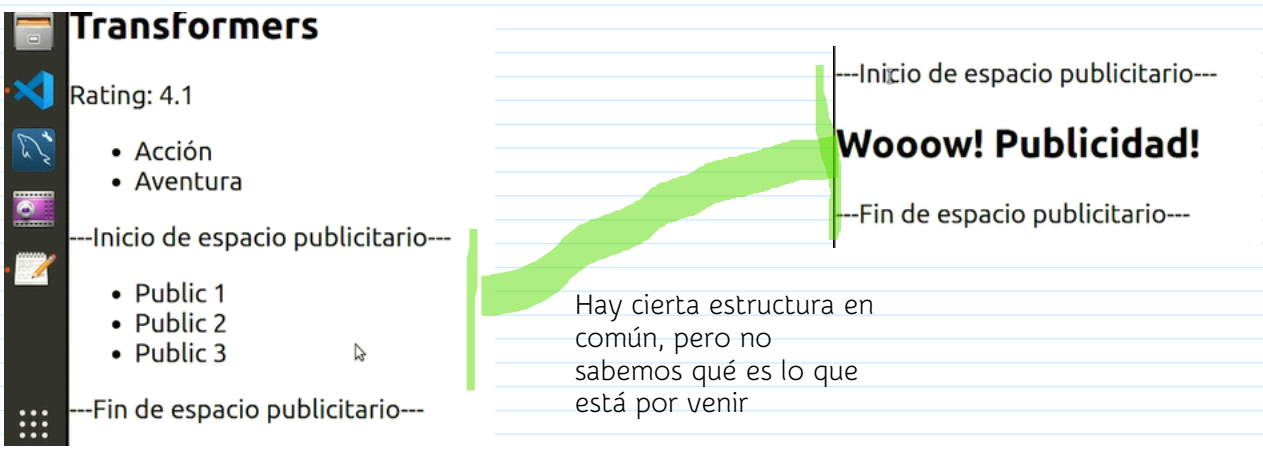


Decidió abrir y cerrar la etiqueta porque dentro de estos vamos a definir qué es lo que va en los children



La primera vez el props.children sólo va a tener el h2, ya en el segunda etiqueta de publicidad el children solo tendrá el

Hace que nuestro componente sea más reutilizable, ya que todo lo que pongamos en la etiqueta de apertura y cierre, el componente cambia radicalmente. Todo esto es cuando no sabemos que hay dentro de nuestro componente y asegurar su flexibilidad.

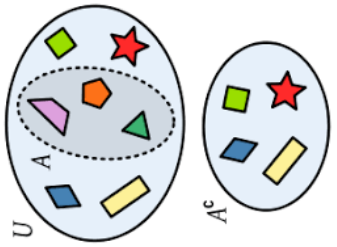


Hay cierta estructura en común, pero no sabemos qué es lo que está por venir

- Esto de children hace que tus componentes sean bloques de código muy reutilzables

Mini resumen

Los componentes children son aquellos elementos que son pasados entre las etiquetas de un componente padre




```
function Saludo(props) {
  return (
          <div>
            <h1>{ props.titulo }</h1>
            <p>{ props.mensaje }</p>
          { props.children }
          </div>
    );
}
```

Con esto le indicamos/aclarando al componente el dónde debería imprimir a los elementos/componentes hijos, si es que los hubiese

Children es una propiedad propia del objeto literal **props**. Y haciendo uso de esta propiedad podemos traer a todos los hijos que definamos en el componente padre pa

```
<main>
    <Saludo>
        // definimos el contenido A
    </Saludo>
<Saludo>
        // definimos el contenido B
    </Saludo>
</main>
```

Tenemos que utilizar o llamar a los componentes que creamos con el siguiente formato para poder agregar todo el contenido que haga falta

¿Cuándo usamos un Children?

Cuando no sabemos que es lo que puede haber dentro de un componente padre. De esta manera (con los children) aseguramos que el componente sea flexible y reutilizable

1 Si tuvieras que explicar la funcionalidad de los children, dirías que:

- Los children permiten que los componentes tengan estado.
- Los children permiten enviar información a los componentes.
- **Los children permiten enviar y renderizar un componente o HTML dentro de otro.**

2 ¿Es verdadera la siguiente afirmación?:
Cuando definimos que un componente va a recibir elementos o componentes por children, estos se pueden imprimir en cualquier lugar del componente padre

- Sí, es verdadera.
- **No, es falsa.**

3 ¿Cómo hacemos para declarar que un componente puede recibir elementos por children?

```
function Componente(props){
 return (
   <div>
     <h1>Hola mundo!</h1>
     {props.children}
   </div>
 )
}
```

La afirmación es falsa ya que los children siempre se van a renderizar en aquel lugar del componente padre donde declaramos que podía llegar a recibir elementos o componentes por children.

Styling

sábado, 5 de marzo de 2022 19:06

- Muy lindo y todo, pero que pasó con el CSS?

React ahora propone componetizar todos nuestros sitios, ¿Será que el css tambien lo componetizaremos?

A medida que componetizamos conseguimos dividir el código de la aplicación en piezas menores, con menor complejidad, lo que seguramente sea beneficioso.

Dentro del universo de React, ¿dónde encaja CSS?

Toda la aplicación de React se renderiza en un archivo que se encuentra en la carpeta public, llamado **index.html**

Y acá podríamos vincular nuestra propia hoja de estilo, siempre y cuando el archivo en cuestión permanezca/se encuentre dentro de la carpeta **public**

Una vez que tengas la hoja de estilos vinculada, alcanza con seguir el formato tradicional de trabajo de CSS. Y siempre teniendo en cuenta que si quieres aplicar una clase dentro de un componente hay que usar el atributo **className** en lugar del atributo class

React también permite tener una hoja de estilos por componente

Esto es muy potente, ya que ahora podemos separar la presentación visual del componente en distintos archivos .css. Ahora podemos trabajar de manera controlada, ya que una hoja de estilos pertenece a un solo componente y no a toda la aplicación

Hay que aclarar que esto en ningún momento te va a permitir usar los mismos nombres de clases entre los componentes, ya que el principio de CSS se sigue manteniendo. Es importante, q dependiendo el componente, tengas cuidado con los nombres de las clases para que entre componentes no se pisen sus reglas de estilo

¿Cómo vinculamos una hoja de estilos con un componente?

```
1  import './EstilosDelComponente.css';
2
```

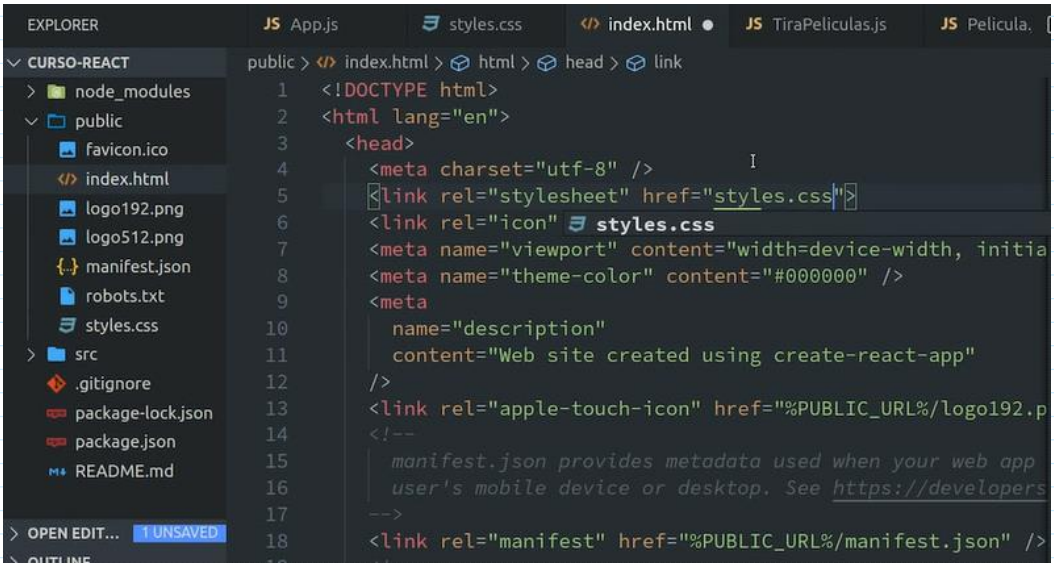
Hacemos referencia a la hoja de estilos que queremos importar y hacer uso de la sentencia **import**

No hace falta darle un alias a lo que estas importando, ya que en este caso sólo queres traer toda la hoja de estilo y que la misma esté disponible para el componente y ya

Luego seguiria usar las clases, id, o demás selectores que hayas definido en la hoja de estilos

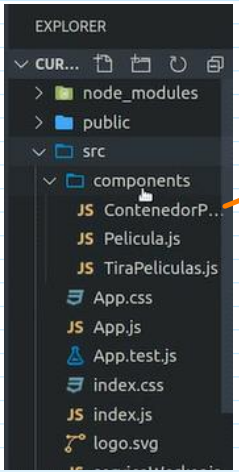
Ejemplo práctico

Vinculación del css con html: [1ra variante]



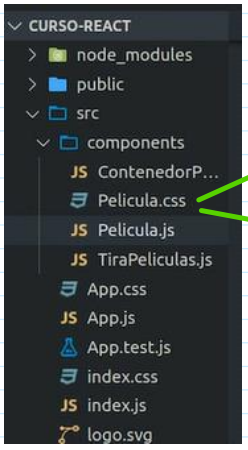
Es bueno para poner estilos generales de mi sitio

Otra variante, modularizando el CSS



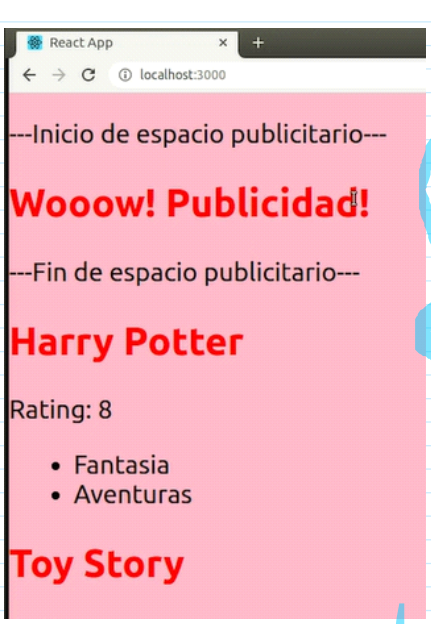
Podemos modularizar al componente al mínimo detalle, incorporando un archivo css para c/u

Lo ideal sería tener subcarpetas para cada componente, en la que dentro de la misma estaría el componente en si y el archivo de estilos



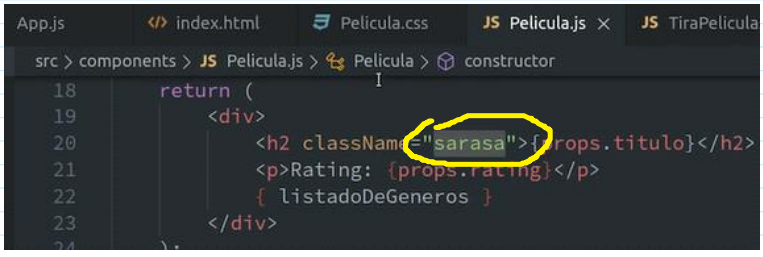
Lo vamos a utilizar para editar el código que tenemos en Pelicula.js

Aunque se llame igual, termina afectando a toda la página

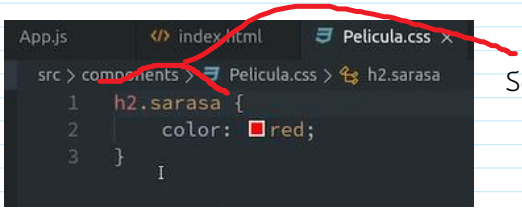


Hay un problema, cambiaron los h2 de toda la página de todos los componentes en general

Solución



★ Vimos como integrar estilos en React, de forma modularizada y super escalable



Selector más específico

- Trabajar con css dentro de los componente de React es izi.
  Ahora hay que darle css a nuestros componentes

Paso 1: Importar el archivo CSS al componente.

Podemos vincular el archivo CSS al archivo del componente de la siguiente manera:

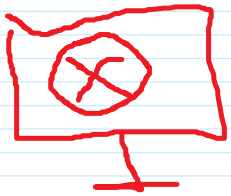
```
import './componentStyling.css
```

Paso 2: Asignarle una clase al componente.

De la misma manera que en HTML y CSS asignamos clases para darle propiedades de CSS a un elemento, en React asignamos clases a los componentes para darles estilo. Esto lo hacemos de la siguiente manera:

```
>_  h2 className="productName"
```

- En HTML y CSS asignabamos clases CSS para darle propiedades CSS a los elementos
- En React asignamos clases a los componentes para darles estilo



Paso 3: No repetir las clases.

Si bien nuestro proyecto está modularizado y parece que en un principio no habría problema en repetir el nombre de las clases, es importante no hacerlo. Esto nos puede traer problemas a la hora de renderizar la totalidad del sitio, ya que se empezarían a pisar nuestras clases entre sí.

1 Al crear un componente, ¿cómo podemos vincular un archivo CSS?

- const css = require("css");
- import css from "./css";
- import "hojaDeEstiloComponente.css";

✓ Muy bien. Al momento de crear nuestros componentes y querer importar el respectivo código CSS, debemos hacerlo con un import y luego indicar el nombre del archivo CSS correspondiente a ese componente.

2 Si queremos asignarle una clase a algún elemento de nuestro componente, ¿cómo deberíamos hacerlo?

- <h1 className = "titulo">Bienvenido al sitio web</h1>

3 Aunque estemos modularizando nuestros estilos, para evitar conflictos al momento de renderizar el sitio, ¿cuál sería una buena práctica para evitar que se pisen nuestras clases entre sí?

- Utilizar className en cada elemento que se le asigne una clase.
- Colocar la misma clase a cada elemento en los distintos componentes.
- No asignarle clases a los elementos.
- No repetir nombre de clases entre hojas de estilos.

✓ Siempre debemos tener muy en cuenta que si colocamos el mismo nombre de clase a los elementos, generaremos conflictos al momento de renderizar el sitio, ya que se pisan las clases entre sí.

React:

- Era para mejorar la performance de nuestras aplicaciones a nivel front-end
- React propone hacer la estructura del front-end completamente con JS utilizando el enfoque de código reutilizable con componentes

Ciclo de vida de un componente?

- Las etapas que pasa el mismo

¿Cuáles son las 3 etapas de un componente en su ciclo de vida?

- Montaje
- Actualización
- Desmontaje

¿Cómo hacemos una llamada a una API desde un componente?

- Conviene hacerlo dentro del método de montaje (DISMOUNT) -> acá actualizamos el estado apenas recibamos la data

¿Qué necesitamos para generar o crear un SPA (single page application?

- El React Router - > React Router DOM

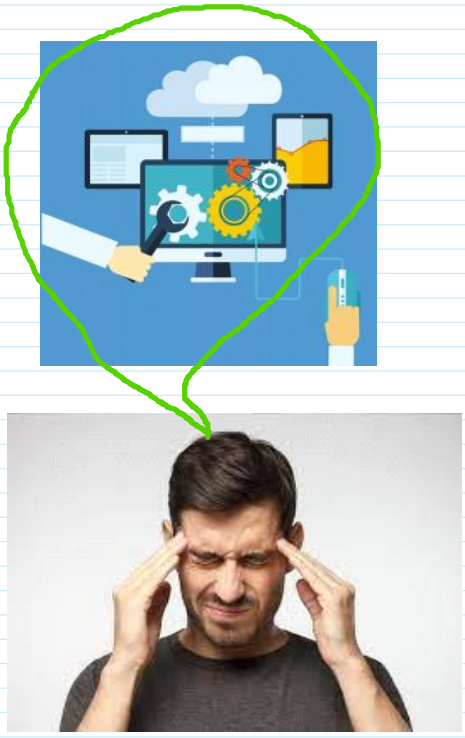
Te paso el resumen porque te conozco 👶: BrowserRouter, Route, Link y Switch son los componentes que te dá React Router para poder generar las rutas de la aplicación y los enlaces de la misma. Mucho cuidado con BrowserRouter🚨, pues lo tenés que usar solamente una vez y ese debe contener a toda la aplicación, sino no funcionará.

Route es una maravilla🎉, se adapta a tus necesidades. Pues para que puedas decirle qué componente querés renderizar, se lo podés indicar como una prop, así: `<Route component={MiComponente} />` o sino como si fuera un children del componente, algo así: `<Route> <MiComponente/> </Route>`.
🚩Ah y no te olvides de la prop **path=""**, esa tiene que ir sí o sí.👀

La última y te abandono (es lo que les digo a todas mis parejas 😅😋😜). Si querés una ruta dinámica, necesitás pasar en la prop path algo así: path="/producto/**:id**", en donde :id es el parámetro que esperás recibir. Luego, dentro del componente que estás renderizando, podés acceder a ese parámetro así: props.match.params.**id**. Fácil y sin complicaciones.💪 Bye, bye.😇

Guía de preguntas para la clase en vivo

sábado, 5 de marzo de 2022 23:15



- Si tuvieras que explicar con tus palabras qué son las props, ¿qué nos podrías decir?
  - Con mis palabras, las props son una especie de parámetro propio del componente y que como valor tiene que ir un objeto literal. Y que dentro de ese objeto literal que mencioné, pueden ir no solo texto fijo, sino tipos de datos de JavaScript como los arrays. Implementando las props, logramos que nuestro componente sea muy reutilizable porque con esto hacemos que su contenido pueda cambiar de acuerdo al contexto en el que se llame al componente en cuestión y definirle las props.
- ¿Cuál sería la diferencia puntual entre las props y los children?
  Con los children podemos agregarle nitro al dinamismo de nuestros componentes, ya que no solo permite que se le puedan mandar estructuras HTML, sino que también que el componente va a estar preparado o listo para recibir hijos que pueden tener una estructura diversa. Con los props solo podíamos mandar texto
- ¿Podemos usar un for dentro de un componente?
  - Nop, no está permitido el uso de ese tipo de iteradores. Tenemos que usar aquellos métodos que iteren sobre un array y que devuelvan un resultado, como por ejemplo:
    - Reduce
    - Filter
    - Map
- ¿Por qué son importantes las key props?
  - Era más que nada para ayudar a React a identificar los elementos. Es más que nada cuando el componente va a renderizar elementos del mismo tipo, necesitaríamos de algún identificador para que React sepa que elementos se agregaron/modificaron/eliminaron
- ¿Cuál es el proceso "natural" y recomendado para poder dar estilos CSS a nuestra aplicación de React?
  - Era simplemente generar los archivos css en la misma carpeta donde se encuentran los componentes e importar en el componente correspondiente el acceso a la hoja de estilos. Algo importante es que por más que hayamos asociado el archivo css al componente, el css continuará afectando a todo mi sitio, por lo que la recomendación es asignarle diferentes nombres de clase a los elementos del componente, todo para que al momento de renderizar no se pisen entre clases

1 ¿Qué son las props de un componente?

- Permiten escribir código modular y empaquetarlo para optimizar el tiempo de carga.
- Conjunto de herramientas adicionales, aplicaciones y librerías que permite a React trabajar como un framework.
- Son entradas de un componente de React. Son información que es pasada desde un componente padre a un componente hijo.

✓ Las props son entradas de un componente de React, que representan información que es pasada desde un componente padre a un componente hijo. Además, pueden recibir propiedades como parámetros para poder insertar valores y eventos en el HTML.

2 ¿Cómo definimos una props dentro del componente?

- Escribiendo una función de JavaScript y, a la misma, le asignamos como parámetro props.

✓ La forma más sencilla de definir una props es escribir una función de JavaScript. El componente no retornará texto fijo, sino que el mismo retornará el contenido de acuerdo al valor de sus propiedades.

3 ¿Qué son los **children** de un componente?

- Proporcionan métodos específicos del DOM que pueden ser utilizados en el nivel más alto de la aplicación.
- Permiten colocar la misma clase a cada elemento en los distintos componentes.
- Es una técnica avanzada en React para el reuso de la lógica de componentes.
- Una propiedad con la que podremos traer a todos los hijos que definamos dentro del componente padre.

Dentro del objeto literal props, existe la propiedad children. Haciendo uso de esta propiedad, podremos traer a todos los hijos que definamos dentro del componente padre.

4 ¿Cómo manejar los children dentro del componente?

- La mejor manera de introducir división de código en tu aplicación es a través de la sintaxis de import() dinámico.
- props.children está disponible en cada componente. Contiene el contenido ubicado entre las etiquetas de apertura y cierre de un componente.

Cuando no sabemos exactamente qué contenido puede llegar a haber dentro de un contenedor padre, usamos los children. De esta manera, estamos configurando un componente flexible y reutilizable.

5 ¿Qué son las proptypes de un componente?

- Son propiedades que permiten la verificación de tipos en las props de un componente.

A medida que tu aplicación crece, podés capturar una gran cantidad de errores con verificación de tipos. React nos permite hacer esto mediante la propiedad especial PropTypes.

6 ¿Para qué sirven las defaulProps de un componente?

- Contienen el contenido ubicado entre las etiquetas de apertura y cierre de un componente.
- Para establecer las props predeterminadas que recibirá un componente.

Las defaultProps pueden ser definidas como una propiedad en la propia clase de componente, que permite establecer las props predeterminadas para la clase. Además, tené también presente que esto se utiliza para props con valor undefined, pero no para props con valor null.