



Front End III

Children + Fragment



Cuando trabajamos con React, podemos crear componentes que actúan como cajas o contenedores genéricos de otros componentes. En tal caso, es probable que deseemos utilizar dichos contenedores en diferentes partes de nuestra aplicación. Frente a ello, el desafío que se nos plantea es que, al momento de crear los contenedores, no conocemos de antemano cuáles serán sus hijos (dado que ello dependerá de cada caso en donde los implementemos).

Para resolver este inconveniente, React nos proporciona un tipo de “prop” especial, conocida como **children**. Dicha propiedad, nos permite pasar elementos hijos directamente al momento de invocar el componente contenedor. De esta manera, podemos anidar arbitrariamente nuestros componentes utilizando contenedores genéricos. Veamos un ejemplo:



```
JS index.js > ...
import React from "react";
import ReactDOM from "react-dom";

const Padre = (props) => {
  return (
    <div
      style={{
        width: "75%",
        background: "#FFEEFD5",
        height: "200px",
        padding: "20px",
      }}
    >
      <h5>Soy un padre</h5>
      {props.children}
    </div>
  );
};

const Hijo = (props) => {
  return (
    <div
      style={{
        width: "50%",
        background: "#FA8072",
        height: "100px",
        padding: "10px",
        color: "white",
        fontSize: "30px",
      }}
    >
      {props.autor}
    </div>
  );
};

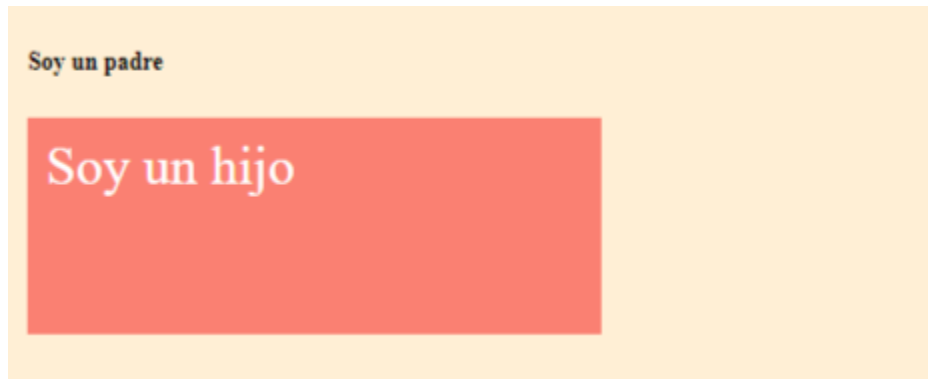
const App = () => {
  return (
    <Padre>
      <Hijo autor="Soy un hijo" />
    </Padre>
  );
};

ReactDOM.render(<App />, document.getElementById("root"));
```



Como puede verse, dentro del componente `<Padre/>`, tenemos una etiqueta `<h5/>` seguida de **props.children**. Esto nos indica que cualquiera sea el componente que anidemos dentro de `<Padre/>`, dicho componente aparecerá a continuación del texto "Soy un padre".

Luego, al utilizar el componente `<Padre/>` en nuestra aplicación, añadimos el componente `<Hijo/>` dentro del mismo. Esto nos da como resultado la siguiente interfaz:



Fragment

Así como deseamos anidar un componente dentro del otro, puede que en determinados casos deseamos retornar varios componentes que se encuentren al mismo nivel. Por ejemplo, podríamos tener un componente `Columns` como el siguiente:

```
const Columns = () => (  
  <td>Hello</td>  
  <td>World</td>  
) ;
```

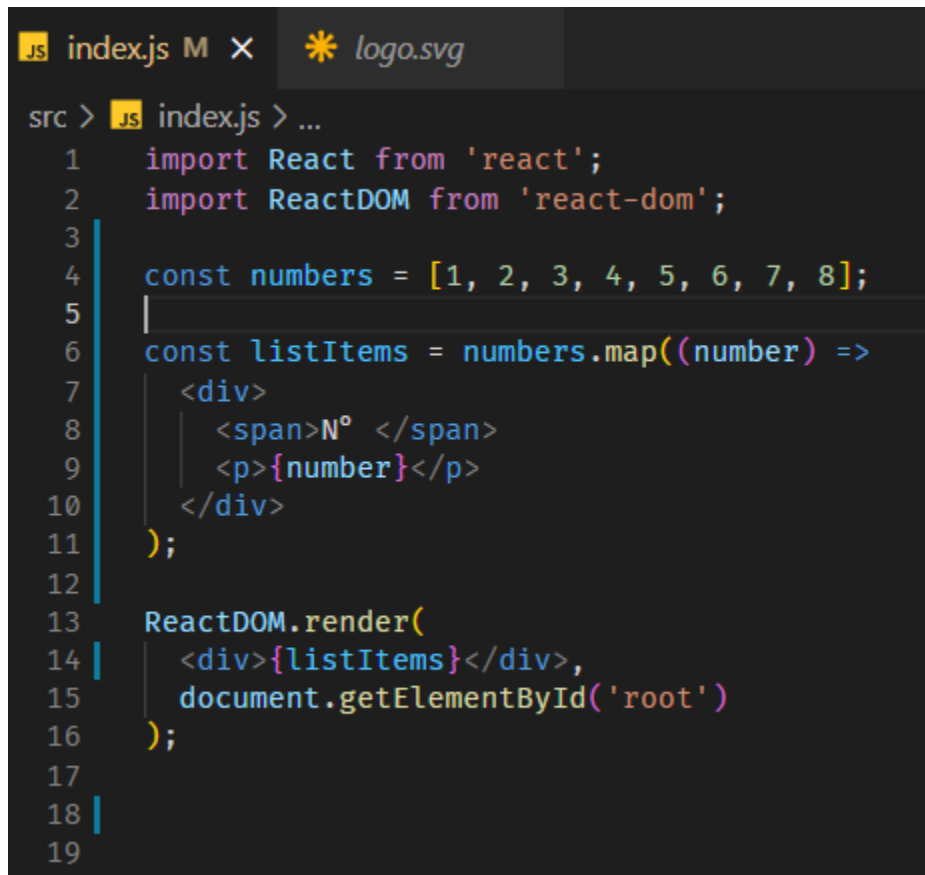
Ahora bien, si intentamos utilizar este componente, obtendremos un error, ya que React solo permite que cada componente retorne un solo elemento JSX como "padre".

Para resolver este problema, simplemente podríamos "agrupar" nuestras etiquetas `<td/>` dentro de un contenedor común, por ejemplo:



```
const Columns = () => (  
  <div>  
    <td>Hello</td>  
    <td>World</td>  
  </div>  
) ;
```

El problema con esto es que en cada uno de estos casos, vamos a crear etiquetas HTML vacías de forma innecesaria. Veamos otro ejemplo. Supongamos que tenemos el siguiente código en donde “agrupamos” las etiquetas `` y `<p/>` dentro de una etiqueta `<div/>`:



```
JS index.js M ✕ * logo.svg  
src > JS index.js > ...  
1 import React from 'react';  
2 import ReactDOM from 'react-dom';  
3  
4 const numbers = [1, 2, 3, 4, 5, 6, 7, 8];  
5  
6 const listItems = numbers.map((number) =>  
7   <div>  
8     <span>Nº </span>  
9     <p>{number}</p>  
10  </div>  
11  );  
12  
13 ReactDOM.render(  
14   <div>{listItems}</div>,  
15   document.getElementById('root')  
16 );  
17  
18  
19
```

Si inspeccionamos el DOM, veremos las siguientes etiquetas:



```
*▼ <body cz-shortcut-listen="true"> == $0
  <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼ <div id="root">
    ▼ <div>
      ▼ <div>
        <span>Nº </span>
        <p>1</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>2</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>3</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>4</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>5</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>6</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>7</p>
      </div>
      ▼ <div>
        <span>Nº </span>
        <p>8</p>
      </div>
    </div>
  </div>
  <!--
```

Como puede verse, se han creado etiquetas `<div>` únicamente para agrupar cada uno de los elementos renderizados. Si bien esto no es un problema grave, agrega contenido innecesario a nuestra aplicación y puede hacer que la misma sea más difícil de inspeccionar en caso que sea necesario realizar dicha tarea.



Para resolver esta situación, React nos brinda la etiqueta `<Fragment>`, la que nos da la posibilidad de agrupar elementos sin crear nodos extras (como vimos anteriormente). Volvamos al ejemplo anterior, pero en este caso reemplazamos la etiqueta `<div>` por el uso de `Fragment`:

```
JS index.js > ...
import React from 'react';
import ReactDOM from 'react-dom';

const numbers = [1, 2, 3, 4, 5, 6, 7, 8];

const listItems = numbers.map((number) =>
  <React.Fragment>
    <span>Nº </span>
    <p>{number}</p>
  </React.Fragment>
);

ReactDOM.render(
  <>{listItems}</>,
  document.getElementById('root')
);
```

Si inspeccionamos nuevamente la aplicación, obtendremos el siguiente resultado:



```
▼<body cz-shortcut-listen="true">
  <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼<div id="root">
    <span>Nº </span>
    <p>1</p>
    <span>Nº </span>
    <p>2</p>
    <span>Nº </span>
    <p>3</p>
    <span>Nº </span>
    <p>4</p>
    <span>Nº </span>
    <p>5</p>
    <span>Nº </span>
    <p>6</p> == $0
    <span>Nº </span>
    <p>7</p>
    <span>Nº </span>
    <p>8</p>
  </div>
  ..
```

Como puede verse, obtenemos un menor número de líneas y un código más legible.

Existen dos maneras de utilizar Fragment:

- La sintaxis que vimos anteriormente, mediante la etiqueta <Fragment> que podemos importar de React, o
- Mediante la sintaxis abreviada, representada por los signos <> y </>.

¡Hasta la próxima!