

APUNTES REACT

Abel Rios - 16/05/2022

Cositas: Añadir a Chrome la extensión "React Developer Tools"

En el Visual Studio añadir la extensión: ES7 React/Redux/React-Native Snippets

Documentación:

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

Para empezar:

```
npx create-react-app my-app  
cd my-app  
npm start
```

Si nos fijamos en el archivo app.js, es básicamente una función (o un archivo) de javascript que retorna html (literal). Eso se denominan archivos .jsx

Estos archivos sólo pueden devolver (return) un elemento padre (un div normalmente, pero puede ser otro elemento, pero sólo uno). Tan grande como tú quieras, pero sólo un elemento.

De tal forma en react se usa la estructura de componentes, que significa que tendremos un archivo que creará el header de la página, otro archivo para el footer y luego archivo/s que sean los componentes del body de las distintas páginas de nuestra web.

Muy importante que nuestros **componentes tengan la primera letra en MAYÚSCULA**, si no la tuvieran en mayúscula, react puede (y suele) confundirlo con una etiqueta html.

Estos componentes los guardamos en una carpeta que llamaremos "components" dentro de nuestra carpeta "src" del proyecto.

Para cada componente crearemos una carpeta (con la primera letra en Mayúscula) y en esa carpeta crearemos un archivo index.js y un archivo con nuestro componente (primera letra en mayúscula). En ese index diremos:

```
export { default } from "../Title";  
// Cuando nuestro app viene a importar a la carpeta Title, lo primero  
que busca es el archivo index  
// de ésta forma, si tenemos varios componentes en la misma carpeta, al  
ir haciendo un "export" en este archivo
```

```
// de todos los componentes de la carpeta, en nuestro app.js sólo  
tenemos que hacer un import y cortito  
// no doscientos imports uno por cada componente  
// O si nuestro componente tiene a su vez más componentes, nuestro  
import no va a ser super largo
```

Para las etiquetas CSS, en React no podemos llamarlas “class” porque es un nombre reservado, para ello usamos la etiqueta “className”. Estos archivos CSS no son necesarios de importarlos en el index del componente, tan solo en el JS del componente.

Volviendo a que react sólo devuelve un elemento padre, en lugar de hacer un div, podemos importar desde react el tipo Fragment:

```
import { Fragment } from "react";
```

Y usar la etiqueta <Fragment> </Fragment>, que funciona como “contenedor” del html que queremos devolver, pero no tiene valor semántico ninguno. No es div ni nada, de hecho si inspeccionamos luego nuestra web, en el html no aparece nada, sí aparece por supuesto el código que hemos generado en react.

Pero ahora empezamos con la **MAGIA**:

Más fácil todavía, el Fragment se puede decir como <> y </>. Y no es necesario ni importar la clase Fragment desde react.

*** Propiedades / Props ***

Forma #1:

Title.js

```
proyecto1 > src > components > Title > JS Title.js > ...  
1  import "./Title.css";  
2  
3  export default function Title(props){  
4  
5      console.log(props);  
6      return(<h1 className="colorFont"> Hola {props.name} </h1>)  
7  
8  }
```

App.js

```
7 |   </>
8 |   <p> Header </p>
9 |   <header className="App-header">
10 |     <Title name="Abel" />
11 |     <img src={logo} className="App-logo" alt="logo" />
12 |     <p>
13 |       Edit <code>src/App.js</code> and save to reload.
```

Web:

Hola Abel



Edit src/App.js and save to reload.

[Learn React](#)

Forma #2 (desestructuración de objeto):

Title.js

```
proyecto1 > src > components > Title > JS Title.js > ...
1  import './Title.css';
2
3  export default function Title(props){
4
5      console.log(props);
6      const {name} = props;
7      return(<h1 className="colorFont"> Hola {name} </h1>)
8
9  }
```

Forma #3 (desestructurando en la misma línea de argumentos de la función)

Title.js

```
proyecto1 > src > components > Title > JS Title.js > ...
1  import './Title.css';
2
3  export default function Title({ name }){
4
5      console.log(name);
6      return(<h1 className="colorFont"> Hola {name} </h1>)
7
8  }
```

*** Reglas No Escritas a la hora de Crear Componentes ***

- Siempre dentro de la carpeta "components"
- Siempre con la primera letra en Mayúscula
- Opcional, crear un archivo "index.js" que importe los componentes
- Hacer el destructuring del objeto props en los paréntesis de la función o en una línea nueva
- Exportar el componente
- Importarlo donde lo queramos usar y añadirlo al archivo jsx

*** Otro ejemplo de props ***

Alumno.js

```
export default function Alumno({alumno}) {  
  
    return(  
        <ul>  
            <li>{alumno.nombre}</li>  
            <li>{alumno.edad}</li>  
            <li>{alumno.email}</li>  
            <li>{alumno.telefono}</li>  
        </ul>  
    )  
}
```

index.js (Alumno)

```
export { default } from "../Alumno"
```

app.jsx

```
import Alumno from '../components/Alumno';  
  
const alumno = { //Estamos suponiendo que ésto nos viene desde la base  
de datos  
    nombre: "Abel Rios",  
    edad:31,  
    telefono: "666666666",  
    email: "abel@mail.com"  
}  
  
function App() {  
    return (  
        <>  
            <p> Header </p>  
            <header className="App-header">  
                <Title name="Abel" />  
                <Alumno alumno={alumno}/>  
                <img src={logo} className="App-logo" alt="logo" />  
            </header>  
        </>  
    )  
}
```

Y ahora vamos a hacer un componente que reciba un array de objetos (alumno)

Alumnos.js

```
import Alumno from "../Alumno/Alumno"

export default function Alumnos({alumnos}) { // donde alumnos va a ser
un array de {alumno}

    return( // react no se lleva guay con los métodos de los array,
para ello usamos un fragment <></>
        <>
            {
                alumnos.map((student) => (<Alumno alumno =
{student}/>))
                // Ponemos el componente Alumno para que nos lo
transforme en el ul + los li
                // como hemos definido en el componente Alumno
            }
        </>
    )
}
```

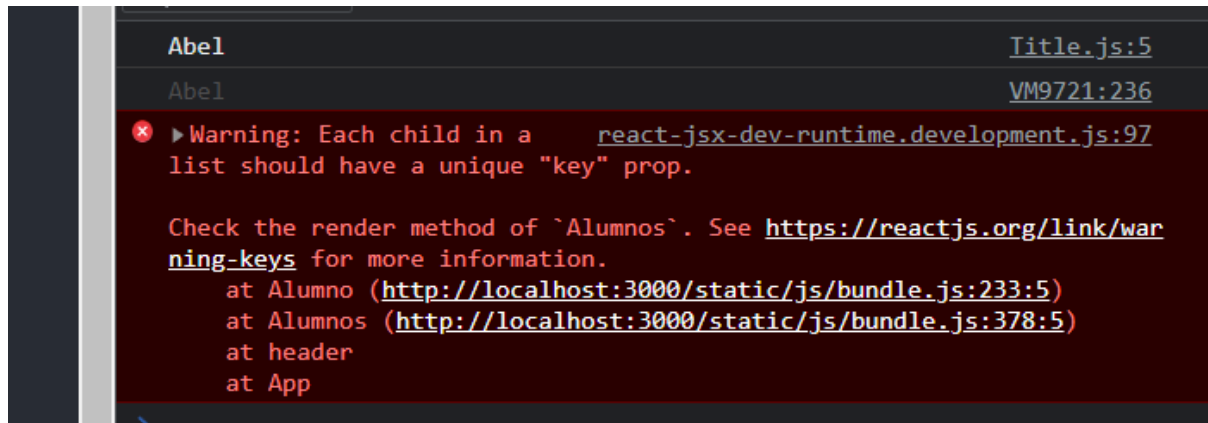
index.js (Alumnos)

```
export { default } from "../Alumnos"
```

app.jsx

```
4   import Alumno from './components/Alumno';
5   import Alumnos from './components/Alumnos';
6
7   const alumnos = [
8     {
9       nombre: "Abel Rios",
10      edad:31,
11      telefono: "666666666",
12      email: "abel@mail.com"
13    },
14    {
15      nombre: "Abel Rios",
16      edad:31,
17      telefono: "666666666",
18      email: "abel@mail.com"
19    },
20    {
21      nombre: "Abel Rios",
22      edad:31,
23      telefono: "666666666",
24      email: "abel@mail.com"
25    },
26    {
31  }
32  ]
33
34  const name = "Abel";
35
36  function App() {
37    return (
38      <>
39        <p> Header </p>
40        <header className="App-header">
41          <Title name = {name}/>
42          <Alumnos alumnos={alumnos}/>
43          <img src={logo} className="App-logo" alt="logo" />
```

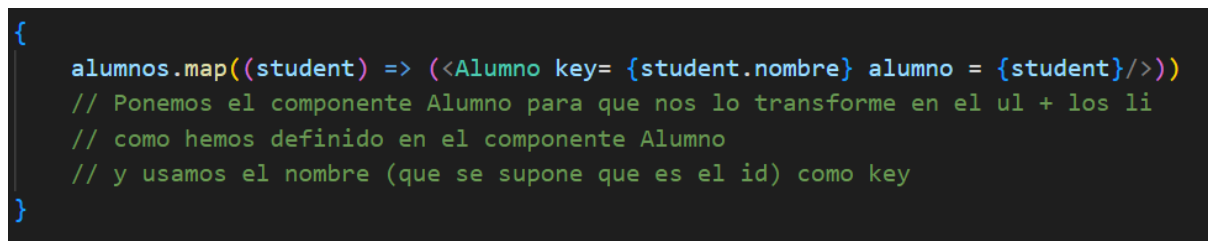
esto nos va a dar un “error” en la consola:



```
Abel Title.js:5
Abel VM9721:236
Warning: Each child in a list should have a unique "key" prop.
Check the render method of `Alumnos`. See https://reactjs.org/link/warning-keys for more information.
    at Alumno (http://localhost:3000/static/js/bundle.js:233:5)
    at Alumnos (http://localhost:3000/static/js/bundle.js:378:5)
    at header
    at App
```

porque estamos creando hijos que no tienen una “key”, para ello ahora vamos a hacer:

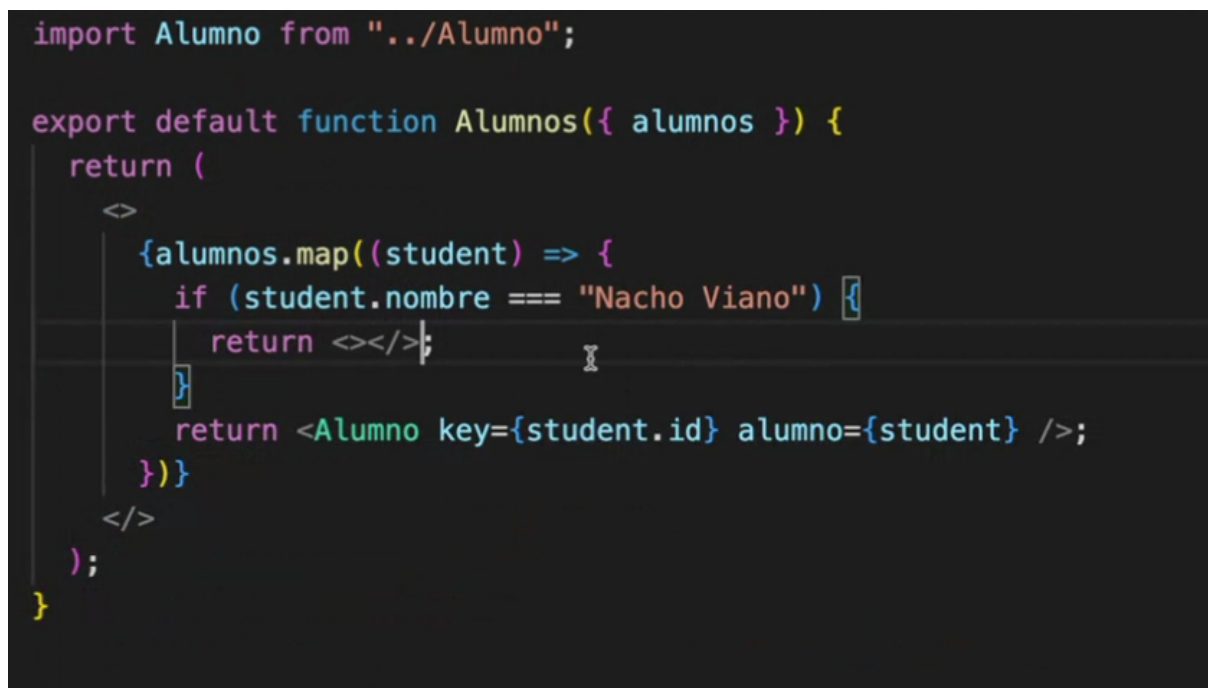
Alumnos.js



```
{
  alumnos.map((student) => (<Alumno key= {student.nombre} alumno = {student}/>))
  // Ponemos el componente Alumno para que nos lo transforme en el ul + los li
  // como hemos definido en el componente Alumno
  // y usamos el nombre (que se supone que es el id) como key
}
```

Aunque lo óptimo es que cada alumno tuviera un id (único del alumno)

Dentro del map también podemos implementar funciones lógicas, ejemplo:



```
import Alumno from "../Alumno";

export default function Alumnos({ alumnos }) {
  return (
    <>
    {alumnos.map((student) => {
      if (student.nombre === "Nacho Viano") {
        return <</>;
      }
      return <Alumno key={student.id} alumno={student} />;
    })}
    </>
  );
}
```

Que nos devolvería “nada” si el nombre del estudiante es Nacho Viano.

O añadir un operador ternario:

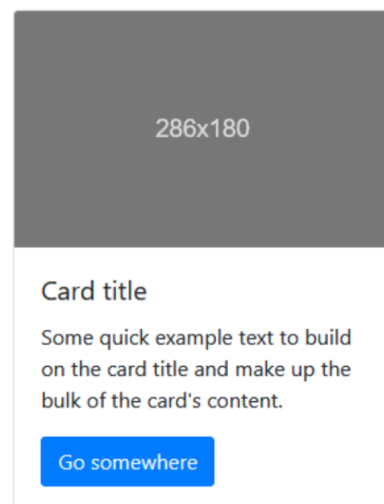
```
1  import Alumno from "../Alumno";
2
3  export default function Alumnos({ alumnos }) {
4    return (
5      <>
6        {alumnos ? (
7          alumnos.map((student) => <Alumno key={student.id} alumno={student} />))
8        } : (
9          <h1>No hay alumnos</h1>
10         )}
11      </>
12    );
13  }
14
```

EJERCICIO

Ejercicio 1

Vamos a crear nuestro primer componente

1. Inicializaremos el proyecto con *create-react-app*.
2. Crearemos el componente Card, con un aspecto similar al de la imagen. Lo haremos directamente en *App.js*.
3. Lo utilizaremos dentro del componente principal App y comprobaremos que se muestra correctamente.
4. Refactorizaremos nuestro componente para que esté en un archivo independiente dentro de una carpeta llamada "components".
5. Haremos que reciba por props la siguiente información: URL de la imagen, título, párrafo, enlace del botón y texto del botón.



Para este ejercicio vamos a importar Bootstrap en un proyecto de React.

<https://www.npmjs.com/package/bootstrap>

"npm i bootstrap"

y en el index importamos:

```
import "bootstrap/dist/css/bootstrap.min.css"
```

