



## **PARTE VII: contexto de REACT**

# INTRODUCCIÓN

En este documento veremos:

- Definición de contexto
- Crear un contexto y como pasar datos entre componentes utilizando el mismo
- Trabajar de forma intensiva con componentes

# INTRODUCCIÓN

## Creación del proyecto

Crearemos un nuevo proyecto desde el cmd o el terminal de VSC:

`create-react-app contexto`

## Preparar el entorno

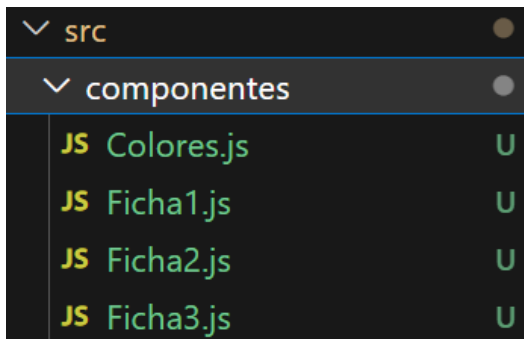
Vamos a editar el fichero **src/App.css** y borraremos el contenido

Editamos el fichero **src/App.js** y vaciamos el contenido del **return** de la función **App()**

# CREAR LOS COMPONENTES

## Crear componentes

Para este ejercicio crearemos, a parte del componente principal, cuatro componentes adicionales que llamaremos **Ficha1.js**, **Ficha2.js**, **Ficha3.js** y **Colores.js** dentro de una carpeta **src/componentes**



### Tandy New Brain FICHA 1

Curiosa máquina que se utilizó en muchos institutos para las prácticas de los alumnos en la recién nacida asignatura de informática.

### Sinclair ZX Spectrum

La más maravillosa máquina que ha existido jamás en el mundo de la microinformática. Gracias a ella existen hoy miles de programadores



### FICHA 2



### Toshiba MSX FICHA 3

Original ordenador que incorporaba el que pretendía ser un estándar en cuanto a funcionamiento: el sistema MSX (preludio del que fuera posteriormente Microsoft)



# CREAR LOS COMPONENTES

## Objetivo del ejercicio

El objetivo del ejercicio es comprender como funciona el contexto de React. Gracias al contexto podemos pasar valores de variables de un componente a otro sin utilizar las propiedades que vimos en otro documento.

Cuando no tenemos demasiados componentes hijos con el uso de las propiedades es suficiente, pero cuando tenemos muchos componentes: hermanos, hijos, nietos, etc. necesitamos un sistema que nos permita compartir los datos comunes entre todos los componentes y este espacio común es el contexto

# CREAR LOS COMPONENTES

## Objetivo del ejercicio

Con este ejercicio cuando pulsemos sobre una de las cajas de colores del componente **Colores** trasladaremos el color de fondo de dicha caja a los tres componentes hermanos **Ficha1** a **Ficha3**



### Tandy New Brain

Curiosa máquina que se utilizó en muchos institutos para las prácticas de los alumnos en la recién nacida asignatura de informática.

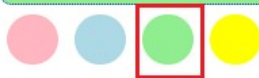
### Sinclair ZX Spectrum

La más maravillosa máquina que ha existido jamás en el mundo de la microinformática. Gracias a ella existen hoy miles de programadores



### Toshiba MSX

Original ordenador que incorporaba el que pretendía ser un estándar en cuanto a funcionamiento: el sistema MSX (preludio del que fuera posteriormente Microsoft)



# CREAR LOS COMPONENTES

## Estructura componentes fichaX.js

Los tres componentes ficha tendrán una estructura similar a la siguiente:

```
<div className='ficha'>  
  <img src={imagen} alt="New Brain" />  
  <h3>Tandy New Brain</h3>  
  <p>Curiosa máquina que se utilizó en muchos institutos para las prácticas  
de los alumnos en la recién nacida asignatura de informática.</p>  
</div>
```

NOTA: Para importar la imagen de cada uno de los tres componentes utilizaremos **require** dentro de la función del componente

**const imagen = require('../img/new\_brain.png');** → para el componente Ficha1

**const imagen = require('../img/sinclair\_zx\_spectrum.png');** → para el componente Ficha2

**const imagen = require('../img/toshiba\_msx.png');** → para el componente Ficha3

# CREAR LOS COMPONENTES

## CSS componentes ficha en App.css

Añadiremos algo de css para los tres componentes ficha. Puesto que no hemos creado ficheros específicos de css para cada componente lo añadiremos en el componente principal dentro del fichero **App.css**

```
.ficha {  
  width: 33%;  
  padding: 15px;  
  border-radius: 8px;  
  border: 1px dotted blue;  
}  
.ficha img {width: 100%;}
```



# CREAR LOS COMPONENTES

## Estructura componentes Colores.js

En este componente incorporamos cuatro cajas en donde, más adelante, añadiremos un evento **onClick** a cada una para permitir el cambio de color de fondo de los tres componentes anteriores

```
<div className='colores'>  
  <div className='color'></div>  
  <div className='color'></div>  
  <div className='color'></div>  
  <div className='color'></div>  
</div>
```

# CREAR LOS COMPONENTES

## CSS componente Colores

Añadiremos css para que las cajas tengan aspecto de botón y cada una de ellas con un color distinto. Puesto que no hemos creado fichero específico de css para este componente lo añadiremos en el componente principal dentro del fichero **App.css**

```
.colores {width: 50%; margin:auto;}  
.color {  
  width: 30px;  
  height: 30px;  
  border-radius: 50%;  
  margin:5px;  
  display: inline-block;  
  cursor: pointer;  
}
```

Para que cada caja tenga un color:

```
.color:nth-child(1) {  
  background-color: lightpink;  
}  
.color:nth-child(2) {  
  background-color: lightblue;  
}  
.color:nth-child(3) {  
  background-color: lightgreen;  
}  
.color:nth-child(4) {  
  background-color: yellow;  
}
```



# CREAR LOS COMPONENTES

## Estructura del componente principal App.js

En el componente principal inyectaremos los cuatro componentes que hemos creado

```
<>
  <div className="App">
    <Ficha1 />
    <Ficha2 />
    <Ficha3 />
  </div>
  <Colores />
</>
```

NOTA 1: Necesitamos la etiqueta vacía `<></>` para que no quede ningún componente fuera de una caja contenedora

NOTA 2: No olvidar importar los componentes con: `import Ficha1 from './componentes/Ficha1';`

# CREAR LOS COMPONENTES

## CSS del componente principal App.js

Y añadiremos también un poco de css para que los tres componentes de las fichas aparezcan situados uno al lado del otro utilizando flex

```
* {box-sizing: border-box;}
```

```
.App {  
  width: 50%;  
  margin:auto;  
  display: flex;  
  justify-content: space-between;  
}
```

# CREAR LOS COMPONENTES

## Estructura del ejercicio

Una vez creados los componentes y el componente principal deberíamos tener lo siguiente (o parecido):



### Tandy New Brain

Curiosa máquina que se utilizó en muchos institutos para las prácticas de los alumnos en la recién nacida asignatura de informática.

### Sinclair ZX Spectrum

La más maravillosa máquina que ha existido jamás en el mundo de la microinformática. Gracias a ella existen hoy miles de programadores



### Toshiba MSX

Original ordenador que incorporaba el que pretendía ser un estándar en cuanto a funcionamiento: el sistema MSX (preludio del que fuera posteriormente Microsoft)

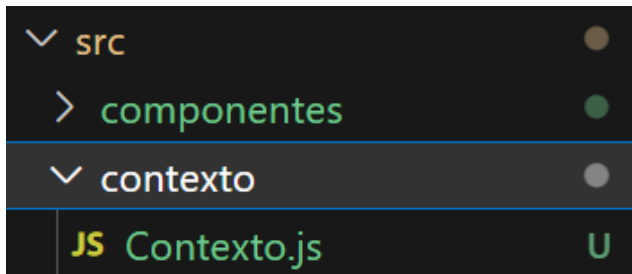


## CREAR EL CONTEXTO

### Crear componente especial con el contexto

El siguiente paso es crear un componente especial que utilizaremos como **contexto**, es decir, el espacio común donde guardaremos toda la información que tendremos que compartir entre todos los componentes y que, en este ejercicio, únicamente será la variable que contenga el color de fondo seleccionado en el componente **Colores** para enviarla a los tres componentes **Ficha**

Creamos dentro de la carpeta **src** una carpeta **contexto** con un archivo **Contexto.js**



## CREAR EL CONTEXTO

### Crear componente especial con el contexto

Editamos el archivo creado y añadiremos lo siguiente:

- Crear una constante con el nombre que queremos asignar al contexto (podemos tener varios contextos cada uno con un nombre distinto)

```
export const Contexto = createContext();
```

- Puesto que `createContext()` pertenece a React lo tendremos que importar:

```
import React, { createContext } from 'react'
```

- Crearemos otra constante con la función que necesitaremos para poder acceder al contexto en cualquier componente y poder recuperar los datos guardados en él:

```
export const Datos = () => { ... }
```

## CREAR EL CONTEXTO

### Indicar qué componentes van a utilizar el contexto creado

Necesitamos indicar que componentes de los que tengamos en la aplicación, van a utilizar el contexto que acabamos de crear para compartir datos.

En nuestro caso son los cuatro componentes que tenemos inyectados en **App.js**, por lo tanto editaremos este fichero e indicamos lo siguiente

```
<>
```

```
<Datos>
```

```
<div className="App">
```

```
<Ficha1 />
```

```
... / ...
```

```
</div>
```

```
<Colores />
```

```
</Datos>
```

```
</>
```

El componente de contexto **Datos** engloba al resto de componentes

NOTA: No olvidar importar Datos:

```
import { Datos } from './contexto/Contexto';
```



## CREAR EL CONTEXTO

### Indicar qué componentes van a utilizar el contexto creado

Ahora vamos a editar de nuevo el fichero **Contexto.js** para completar la función **Datos** en donde indicaremos que variables se van a compartir y que componentes podrán compartirlas o acceder a ellas, es decir, tenemos que indicar a que componentes tiene que dar servicio esta función.

Puesto que tiene que dar servicio a todos los componentes que se encuentran dentro de `<Datos>` en el fichero **App.js**, indicaremos que se encargue React de buscar todos los componentes hijos:

```
export const Datos = ({ children }) => { ... }
```

## CREAR EL CONTEXTO

### Indicar qué variables se van a compartir

El siguiente paso es indicar en el fichero **contexto.js**, dentro de la función **Datos**, que variables se van a compartir y, para ello, tenemos que indicar lo siguiente:

```
export const Datos = ({ children }) => {  
  return (  
    <Contexto.Provider>  
      { children }  
    </Contexto.Provider>  
  )  
}
```

Lo que hará React es substituir la etiqueta **<Datos>** que tenemos en **App.js** por el **return** que hemos indicado en la función **Datos** de **contexto.js**

## CREAR EL CONTEXTO

### Indicar qué variables se van a compartir

¿Y qué es el **provider**?

Cada objeto **Context** viene con un componente **Provider** de React que permite que los componentes que lo consuman se suscriban a los cambios del contexto (detecten los cambios de estado, es decir, los cambios en el valor de las variables que le indiquemos).

El componente **Provider** acepta una propiedad **value**, que añadiremos más adelante, que se pasará a los componentes consumidores que son descendientes de este **Provider** y que, en este ejemplo, son todos los componentes hijos (por esto indicamos **{children}** dentro del **Provider**), es decir: **Ficha1**, **Ficha2**, **Ficha3** y **Colores**

## CREAR EL CONTEXTO

### Indicar qué variables se van a compartir

¿Y qué variable necesitamos compartir?

Pues la variable donde guardaremos el color del fondo a asignar a los componentes y cuyo valor se obtendrá del componente **Colores**.

Recordad que en React no podemos asignar o modificar valores a las variables de forma directa, por lo que necesitaremos el hook **useState**:

```
export const Datos = ({ children }) => {  
  const [color, setColor] = useState(); //recordad importarlo  
  return (  
    <Contexto.Provider value={{color, setColor}}>  
      { children }  
    </Contexto.Provider>  
  )  
}
```

NOTA: Observad como añadimos lo que vamos a compartir dentro del atributo **value** del **Provider** en forma de objeto con un atributo **color** y un método **setColor**

## CAMBIAR EL COLOR DE FONDO

### Activar los eventos de cambio de color en Colores.js

Ya tenemos todo el entorno y los componentes creados. Ahora solo falta añadir los eventos **onClick** en las cajas del componente **colores.js** y asociarles una función que permita asignar el valor del color que después tendremos que recoger en cada uno de los tres componentes

#### Ficha

```
<div className='color' onClick={()=>setColor('lightpink')}></div>  
<div className='color' onClick={()=>setColor('lightblue')}></div>  
<div className='color' onClick={()=>setColor('lightgreen')}></div>  
<div className='color' onClick={()=>setColor('yellow')}></div>
```

# CAMBIAR EL COLOR DE FONDO

## Activar los eventos de cambio de color en Colores.js

¿Y de donde obtenemos **setColor**?

Pues el método que hemos indicado en **contexto.js** dentro de la propiedad **value** del Provider como uno de los elementos que compartiremos entre el resto de componentes.

Por lo tanto tendremos que definirlo dentro de **colores.js** de la siguiente forma:

```
const {setColor} = useContext(Contexto);
```

NOTA 1: Como podemos tener más de un contexto tendremos que indicar en **useContext()** el nombre del que queremos utilizar (en nuestro caso **Contexto** que es el único que hemos creado)

NOTA 2: Observad como desestructuramos el método **setColor** ya que el el archivo de contexto estamos compartiendo un objeto con un atributo **color** y un método **setColor** y, en este componente, solo necesitamos el método **setColor**

## CAMBIAR EL COLOR DE FONDO

**Activar los eventos de cambio de color en Colores.js**

E importar tanto el Contexto como el método **useContext**:

```
import React, { useContext } from 'react';
```

```
import { Contexto } from '../contexto/Contexto';
```

## CAMBIAR EL COLOR DE FONDO

### Asignar el color seleccionado a los componentes

Lo único que nos falta es asignar el color informado en el componente **colores.js** y que tendremos en el contexto, en cada uno de los componentes de las fichas. Ejemplo para **Ficha1.js**:

- Importamos el contexto y el correspondiente método:

```
import React, { useContext } from 'react';  
import { Contexto } from '../contexto/Contexto';
```

- Definimos la variable a utilizar para el color

```
const {color} = useContext(Contexto);
```

- Y utilizando estilos en línea (sí, ya sé que no son los más adecuados) cambiamos el color de fondo de la ficha:

```
<div className='ficha' style={{background: color}}>
```



# ACTIVIDAD 7

## ACTIVIDAD

Vamos a realizar una actividad compleja en la que pondremos en práctica todo lo que hemos visto en este y en los anteriores documentos del curso:

Crearemos una aplicación para consultar las fichas de tres profesoras de React en donde mostraremos su nombre, ubicación y fotografía a parte de los datos generales del curso y la dedicación diaria

Cada profesora imparte el curso en un idioma distinto que podemos seleccionar en un componente con las opciones: español. Inglés o francés de forma que, al pulsar sobre cada una de las imágenes de cambio de idioma, se mostrará la ficha que corresponda a la profesora que imparte el curso en el idioma seleccionado

# ACTIVIDAD

Veamos gráficamente un ejemplo:




**Aprenda React intensivamente con una profesora nativa**  
2 semanas. Una profesora sólo para ti (12h/día)

Profesora

Lugar


**Profesora:**  
  
Marta Ríos



**Learn React intensively with a native teacher**  
2 weeks. A teacher just for you (12h/day)

Professor

Location

**Professor:**  
  
Grace Trembley



**Apprenez React intensément avec un professeur natif**  
2 semaines. Un professeur rien que pour vous (12h/jour)

Professeur

Emplacement

**Professeur:**  
  
Aimée Mathieu


## ACTIVIDAD

Además, en cada ficha, podemos seleccionar mostrar la foto de la profesora o la ubicación física de las clases pulsando el botón correspondiente:

Profesora

Lugar

**Profesora:**



Marta Ríos

Profesora

Lugar

**Lugar:**

48 St Laurent Boulevard, Montreal, Canadá

# ACTIVIDAD

## Partes de la actividad:

1. Creación de los componentes
2. Creación del contexto
3. Creación de la operativa de carga dinámica de componentes
4. Creación de la operativa de selección de idioma

# ACTIVIDAD

## Actividad 7.1: Creación de los componentes

Tendremos que crear los componentes que podemos ver en las siguientes imágenes



# ACTIVIDAD

## Actividad 7.1: Creación de los componentes

Relación de componentes:

- Componente principal: **App.jsx**
- Componente para el cambio de idioma: **componentes/Idiomas.jsx**
- Componente para el contenido (ficha de la profesora):  
**componentes/contenido.jsx**
  - Componente para la foto y nombre de la profesora y que inyectaremos dentro del componente contenido: **componentes/profesora.jsx**
  - Componente para la ubicación de la profesora y que inyectaremos dentro del componente contenido: **componentes/ubicacion.jsx**

Para facilitar el desarrollo, todo el CSS que necesitemos lo ubicaremos dentro del archivo del componente principal **App.css**

# ACTIVIDAD

## Actividad 7.1: Creación de los componentes

Además crearemos dentro de la carpeta de **componentes** un fichero adicional **profesoras.js** con el array que necesitaremos con los datos de cada una de las profesoras (y que se adjunta como recursos del ejercicio)

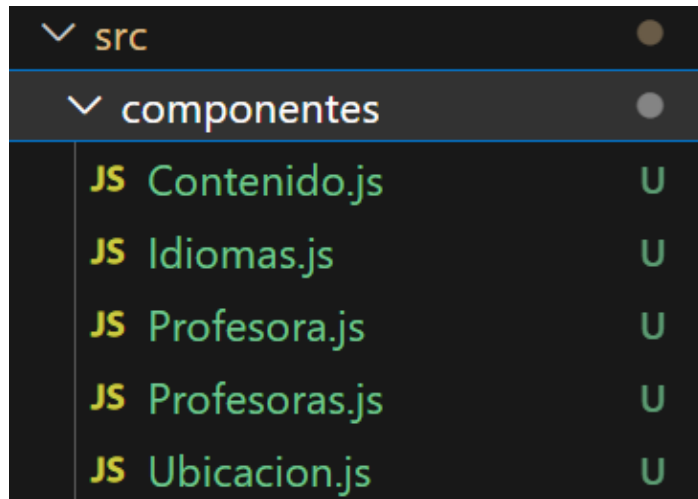
```
const profesoras=[{ ... }, { ... }, { ... }]  
export default profesoras;
```



# ACTIVIDAD

## Actividad 7.1: Creación de los componentes

Tendremos una estructura de carpeta similar a la siguiente:



## ACTIVIDAD

### Actividad 7.1: Estructura del componente principal App.jsx

En el componente principal inyectaremos los componentes **Idiomas** y **Contenido**:

```
<div className='App'>  
  <div className='banderas'>  
    <Idiomas />  
  </div>  
  <div className='contenido'>  
    <Contenido />  
  </div>  
</div>
```

Y un poco de css en **App.css**:

```
* {box-sizing: border-box;}  
.App { width: 50%;margin:auto;padding: 20px;border: 1px solid grey;}
```

NOTA: Recordad incorporar los correspondientes **import**

## ACTIVIDAD

### Actividad 7.1: Estructura del componente Idiomas.jsx

En este componente incorporaremos las cajas con las tres banderas de cambio de idioma

```
<div className='idiomas'>
  <div className='bandera'>
    <img src={banderaEs} alt="bandera"/>
  </div>
  <div className='bandera'>
    <img src={banderaUk} alt="bandera"/>
  </div>
  <div className='bandera'>
    <img src={banderaFr} alt="bandera"/>
  </div>
</div>
```

## ACTIVIDAD

### Actividad 7.1: Estructura del componente Idiomas.jsx

Tendremos que importar con **require** o con **import** las imágenes a utilizar (y que se adjuntan como recursos del ejercicio):

```
const banderaEs = require('../img/spain.jpg')  
const banderaFr = require('../img/francia.png')  
const banderaUk = require('../img/uk.png')
```

Y el css necesario en **App.css**

```
img {width: 100%; vertical-align: top;}  
.idiomas {display: flex; justify-content: center;}  
.bandera {width: 75px; margin: 10px}
```



## ACTIVIDAD

### Actividad 7.1: Estructura del componente Contenido.jsx

En este componente incorporaremos el nombre del curso y disponibilidad, los botones de selección de foto o ubicación e inyectaremos los dos componentes **profesora** y **ubicación**:

```
<>
  <h1>Nombre curso</h1>
  <h2>Disponibilidad</h2>
  <div className='botones'>
    <button>Profesora</button>
    <button>Lugar</button>
  </div>
  <Profesora/>
  <Ubicacion/>
</>
```

NOTA: Posteriormente tendremos que informar de forma dinámica cada uno de los valores a mostrar en la ficha a partir del array **profesoras.js**

# ACTIVIDAD

## Actividad 7.1: Estructura del componente Contenido.jsx

Añadiremos el css necesario

```
.botones {text-align: center;}  
.botones button {  
  width: 150px;  
  padding: 5px;  
  margin: 10px;  
  border-radius: 5px;  
  border: 1px dotted blue;  
  color: white;  
  background-color: brown;  
  font-size: 1.1em;  
}
```



## ACTIVIDAD

### Actividad 7.1: Estructura del componente Profesora.jsx

En este componente que inyectamos dentro de contenido mostraremos la foto y el nombre de la profesora del curso

```
<div className='profesora'>  
  <h2>Profesora:</h2>  
  <div className='foto'>  
    <img src={foto} alt="profesora"/>  
  </div>  
  <div className='nombre'>  
    Nombre profesora  
  </div>  
</div>
```

NOTA: Provisionalmente informaremos una foto cualquiera de las tres:

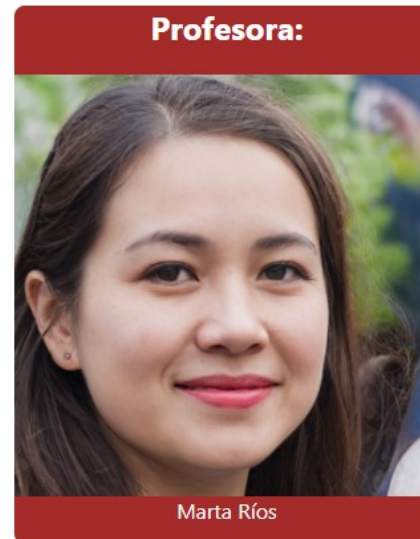
```
const foto = require(`../img/marta.PNG`)
```

# ACTIVIDAD

## Actividad 7.1: Estructura del componente Profesora.jsx

Y añadimos un poco de css en **App.css**

```
.profesora, .ubicacion {  
  width: 320px;  
  text-align: center;  
  border-radius: 8px;  
  background-color: brown;  
  padding-bottom: 15px;  
  color:white;  
  margin:auto;  
}
```



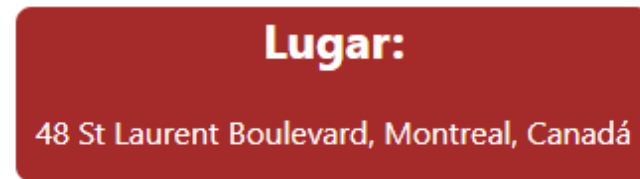


# ACTIVIDAD

## Actividad 7.1: Estructura del componente Ubicacion.jsx

En este componente que inyectamos dentro de contenido mostraremos la dirección donde se realizará el curso

```
<div className='ubicacion'>  
  <h2>Lugar: </h2>  
  <div className='direccion'>  
    Dirección  
  </div>  
</div>
```



Utilizaremos el mismo css que el que hemos asignado al componente **profesora**

## ACTIVIDAD

### Actividad 7.2: Creación del componente de contexto

Vamos a crear los componentes de contexto que utilizaremos para:

- Crear el contexto
- Crear el proveedor de información que utilizará los componentes

Crearemos, por tanto, dos ficheros dentro de una carpeta llamada **src/contexto**:

Fichero **Contexto.jsx** con la creación del contexto

```
import { createContext } from "react";  
const Contexto = createContext();  
export default Contexto;
```

NOTA: Tenemos que exportar la constante **Contexto** para poder utilizarla fuera del archivo contexto.js

## ACTIVIDAD

### Actividad 7.2: Creación del componente de contexto

Fichero **Provider.jsx** con la creación del proveedor de información

```
import { useState } from "react"
import Contexto from './Contexto'
import profesoras from '../componentes/Profesoras'
const Provider = ({children}) => {
  const [idioma, setIdioma] = useState(0);
  const [profesora, setProfesora] = useState(profesoras[0])
  return (
    <Contexto.Provider value={{setIdioma, profesora}}>
      {children}
    </Contexto.Provider>
  )
}
export default Provider;
```

# ACTIVIDAD

## Actividad 7.2: Creación del componente de contexto

### Fichero **Provider.jsx** con la creación del proveedor de información

En este fichero:

- Importamos las clases de react que necesitaremos
- Importamos el array de profesoras
- Definimos la constante **Provider** para el proveedor de información a compartir entre los componentes
- Indicamos que componentes van a compartir la información (todos los hijos)
- Informamos utilizando **useState** la variable **idioma** con el valor por defecto 0 que corresponde al índice 0 del array **profesoras** (idioma castellano)
- Informamos utilizando **useState** la variable **profesora** a partir del índice 0 del array **profesoras** (idioma castellano)
- Indicamos que valor y que método se van a usar en todos los componentes (el método **setIdioma** para modificar la variable **idioma** en el componente **Idioma.jsx** y la variable **profesora** con el objeto que corresponda a la profesora del idioma seleccionado)

## ACTIVIDAD

### Actividad 7.2: Inyectar el proveedor dentro de App.jsx

Tenemos que inyectar el proveedor que hemos creado dentro del componente principal **App.jsx** para indicar que componentes son los que forman parte de él:

```
return (  
  <Provider>  
    <div className='App'>  
      <div className='banderas'>  
        <Idiomas />  
      </div>  
      <div className='contenido'>  
        <Contenido />  
      </div>  
    </div>  
  </Provider>  
)
```

## ACTIVIDAD

### Actividad 7.3: Carga dinámica de componentes

Vamos a realizar ahora que tenemos todos los componentes confeccionados, la carga dinámica de los mismos para que muestren la información que corresponda a cada una de las profesoras.

Primero, para no complicar en exceso el ejercicio, trabajaremos solo con la profesora que ocupa la posición 0 en el array **profesoras** y que corresponde al idioma español

# ACTIVIDAD

## Actividad 7.3: Carga dinámica de componentes

### Componente **Contenido.jsx**

Vamos a editar el componente **Contenido.jsx** para informar los datos que corresponden al nombre del curso y la disponibilidad de la profesora

- Recuperamos la variable de contexto que hemos creado en el fichero **Provider.jsx**:

```
const {profesora} = useContext(Contexto)
```

- Informamos el literal que corresponde al nombre del curso

```
<h1>{profesora.titulo}</h1>
```

- Informamos el literal que corresponde a la disponibilidad

```
<h2>{profesora.texto}</h2>
```

# ACTIVIDAD

## Actividad 7.3: Carga dinámica de componentes

### Componente Contenido.jsx

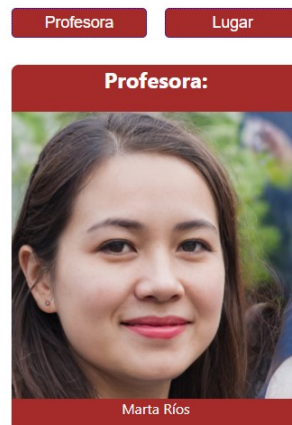
- Y los literales que corresponden a los dos botones:

```
<button>{profesora.boton1}</button>
```

```
<button>{profesora.boton2}</button>
```

**Aprenda React intensivamente con una profesora nativa**

2 semanas. Una profesora sólo para ti (12h/día)





# ACTIVIDAD

## Actividad 7.3: Carga dinámica de componentes

### Componente Contenido.jsx

Ahora lo que tenemos que hacer es habilitar los dos botones que tenemos en este componente de forma que, al pulsar uno u otro, se muestre el componente `<Profesora>` o el componente `<Ubicacion>`

Profesora

Lugar

- Vamos a crear una variable de tipo booleano de forma que, si contiene el valor **true** se muestre el componente **profesora** y, si contiene el valor **false**, se muestre el componente **ubicación**

```
const [profesoraUbicacion, setProfesoraUbicacion] = useState(true);
```

NOTA: Por defecto la informamos a **true** para que se muestre el componente que corresponde a la foto de la profesora

## ACTIVIDAD

### Actividad 7.3: Carga dinámica de componentes

#### Componente Contenido.jsx

- Ahora vamos a activar un evento **onClick** en cada uno de los botones para que ejecuten una función anónima que lo único que va a hacer es cambiar el valor de la variable **profesoraUbicacion**

```
<button onClick={()=>setProfesoraUbicacion(true)}>...</button>
```

```
<button onClick={()=>setProfesoraUbicacion(false)}>...</button>
```

- Ahora lo que nos falta es mostrar un componente u otro de forma condicional:

```
{profesoraUbicacion ? <Profesora/> : null}
```

```
{profesoraUbicacion ? null : <Ubicacion/>}
```

O bien

```
{profesoraUbicacion ? <Profesora/> : <Ubicacion/>}
```

NOTA: Con **true** se mostrará el componente **<Profesora>** y con **false** se mostrará **<Ubicacion>**

## ACTIVIDAD

### Actividad 7.3: Carga dinámica de componentes

#### Componente Contenido.jsx

- Desde este componente podemos enviar el objeto **profesora** que hemos recuperado del contexto a los componentes **<Profesora>** y **<Ubicacion>** utilizando las propiedades de los componentes

```
{componente ?  
  <Profesora profesora={profesora}/> :  
  <Ubicacion profesora={profesora}/>  
}
```

- También podemos cargar los componentes sin enviar parámetros y recuperar luego del contexto el objeto **profesora** en cada uno de ellos

## ACTIVIDAD

### Actividad 7.3: Carga dinámica de componentes

#### Componente Profesora.jsx

Vamos a editar el componente **Profesora.jsx** para informar todos los datos que corresponden a la profesora que corresponda al valor del objeto profesora que informamos por defecto en el fichero de contexto (recordad que por defecto es el 0)

- Si hemos enviado el objeto **profesora** utilizando las propiedades desde el componente Inicio lo recuperamos en los parámetros de la función del componente:

```
const Profesora = ({profesora}) => { ... }
```

- En caso contrario, tendremos que recuperar la variable de contexto donde indicamos la profesora a mostrar y que hemos creado en el fichero **Provider.jsx**:

```
const {profesora} = useContext(Contexto)
```

# ACTIVIDAD

## Actividad 7.3: Carga dinámica de componentes

### Componente Profesora.jsx

- Ya podemos informar de forma dinámica la foto de la profesora que estábamos informando hasta ahora con una por defecto

```
const foto = require(`../img/${profesora.foto}`)
```

- Informamos también de forma dinámica el texto que tiene que aparecer en la cabecera de la foto

```
<h2>{profesora.boton1}</h2>
```

- Y el nombre de la profesora que también estábamos informando con uno por defecto

```
<div className='nombre'>{profesora.nombre}</div>
```

## ACTIVIDAD

### Actividad 7.3: Carga dinámica de componentes

#### Componente **Ubicacion.jsx**

Vamos a editar el componente **Ubicacion.jsx** para informar también todos los datos que corresponden a la profesora

- Si hemos enviado el objeto **profesora** utilizando las propiedades desde el componente Inicio lo recuperamos en los parámetros de la función del componente:

```
const Ubicacion = ({profesora}) => { ... }
```

- En caso contrario recuperamos la variable de contexto que hemos creado en el fichero **Provider.js**:

```
const {profesora} = useContext(Contexto)
```

# ACTIVIDAD

## Actividad 7.3: Carga dinámica de componentes

### Componente Ubicacion.jsx

- Informamos el literal que corresponde al texto 'Lugar'

```
<h2>{profesora.boton2}</h2>
```

- Y la dirección de la profesora que también estábamos informando con uno por defecto

```
<div className='direccion'>{profesora.direccion}</div>
```

**Lugar:**

48 St Laurent Boulevard, Montreal, Canadá

# ACTIVIDAD

## Actividad 7.4: Selección de idioma

### Componente Idiomas.jsx

Por último vamos a modificar el componente de cambio de idioma para poder modificar el valor de la variable de contexto **idioma** y asignarle el valor que corresponda al idioma seleccionado:

- 0 para español
- 1 para inglés
- 2 para francés

Y que corresponden a la posición que ocupa dentro del array los datos de cada una de las tres profesoras



# ACTIVIDAD

## Actividad 7.4: Selección de idioma

### Componente Idiomas.jsx

- Primero tenemos que recuperar el método **setIdioma** desde el contexto

```
const {setIdioma} = useContext(Contexto)
```

- Vamos a asociar un evento **onClick** a cada una de las cajas que contienen las imágenes de las banderas para que ejecuten el método **setIdioma** con el valor que corresponda a cada uno de ellos:

```
<div className='bandera' onClick={()=>setIdioma(0)}> → para español
```

```
<div className='bandera' onClick={()=>setIdioma(1)}> → para inglés
```

```
<div className='bandera' onClick={()=>setIdioma(2)}> → para francés
```

# ACTIVIDAD

## Actividad 7.4: Selección de idioma

### Componente **Provider.jsx**

Obviamente todavía no funciona el ejercicio porque, recordemos, en el fichero **Provider.jsx** tenemos todavía por defecto el idioma 0 para recuperar la profesora del array

```
const [profesora, setProfesora] = useState(profesoras[0])
```

Tenemos que utilizar algún mecanismo que nos permita seleccionar la profesora del array de forma dinámica, es decir, dependiendo del idioma que seleccionemos en el componente **Idiomas.jsx** y que queda guardado en la variable de contexto **idioma**

# ACTIVIDAD

## Actividad 7.4: Selección de idioma

### Componente **Provider.jsx**

¿Y cual es este mecanismo?

Pues el hook **useEffect** que nos permitirá modificar el contenido del objeto **profesora** cuando cambie el contenido de la variable **idioma** (nos subscribimos al cambio de la variable **idioma**):

- Importamos el hook:

```
import { useEffect, useState } from "react"
```

- Incorporamos el hook dentro de **Provider** y utilizamos el método **setProfesora** para cambiar el valor del objeto profesora

```
const [profesora, setProfesora] = useState(profesoras[0])
```

```
useEffect(() => {  
    setProfesora(profesoras[idioma])  
}, [idioma])
```

## ACTIVIDAD

Si todo funciona correctamente veremos como se cambia el idioma y se selecciona la ficha correspondiente a la profesora que imparte el curso en el idioma seleccionado



**Aprenda React intensivamente con una profesora nativa**  
2 semanas. Una profesora sólo para ti (12h/día)

Profesora

Lugar

**Profesora:**  
  
Marta Ríos



**Learn React intensively with a native teacher**  
2 weeks. A teacher just for you (12h/day)

Professor

Location

**Professor:**  
  
Grace Trembley



**Apprenez React intensément avec un professeur natif**  
2 semaines. Un professeur rien que pour vous (12h/jour)

Professeur

Emplacement

**Professeur:**  
  
Aimée Mathieu

## Aviso Legal

Los derechos de propiedad intelectual sobre el presente documento son titularidad de David Alcolea Martinez Administrador, propietario y responsable de [www.alcyon-it.com](http://www.alcyon-it.com) El ejercicio exclusivo de los derechos de reproducción, distribución, comunicación pública y transformación pertenecen a la citada persona.

Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro).

La ley que ampara los derechos de autor establece que: “La introducción de una obra en una base de datos o en una página web accesible a través de Internet constituye un acto de comunicación pública y precisa la autorización expresa del autor”. El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multa, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujesen, plagiaran, distribuyeren o comunicaren públicamente, en todo o en parte, o su transformación, interpretación o ejecución fijada en cualquier tipo de soporte o comunicada a través de cualquier medio.

El usuario que acceda a este documento no puede copiar, modificar, distribuir, transmitir, reproducir, publicar, ceder, vender los elementos anteriormente mencionados o un extracto de los mismos o crear nuevos productos o servicios derivados de la información que contiene.

Cualquier reproducción, transmisión, adaptación, traducción, modificación, comunicación al público, o cualquier otra explotación de todo o parte del contenido de este documento, efectuada de cualquier forma o por cualquier medio, electrónico, mecánico u otro, están estrictamente prohibidos salvo autorización previa por escrito de David Alcolea. El autor de la presente obra podría autorizar a que se reproduzcan sus contenidos en otro sitio web u otro soporte (libro, revista, e-book, etc.) siempre y cuando se produzcan dos condiciones:

1. Se solicite previamente por escrito mediante email o mediante correo ordinario.
2. En caso de aceptación, no se modifiquen los textos y se cite la fuente con absoluta claridad.

David Alcolea  
david-alcolea@alcyon-it.com  
www.alcyon-it.com

