

UT4

Integración de contenido interactivo.

React

Indice

1.- React.....	3
<i>1.1.- Lo necesario para utilizar React.....</i>	<i>3</i>
<i>1.2.- Preparación del entorno.....</i>	<i>4</i>
1.2.1.- ¿Qué es Node.js?.....	4
1.2.2.- ¿Qué es npm?.....	4
1.2.3.- Instalación de Node.js.....	4
<i>1.3.- Primeros pasos en React.....</i>	<i>5</i>
<i>1.4.- Formas de escribir código en React.....</i>	<i>8</i>
<i>1.5.- Variables en React.....</i>	<i>8</i>
<i>1.6.- Componentes en React.....</i>	<i>8</i>
<i>1.7.- Componentes de clase.....</i>	<i>10</i>
<i>1.8.- Componentes simples.....</i>	<i>12</i>
<i>1.9.- Props.....</i>	<i>13</i>
<i>1.10.- State – Permite modificar los datos de un componente.....</i>	<i>18</i>

1.- React

React es una biblioteca de JavaScript para crear interfaces de usuario. Este proyecto es impulsado por Facebook Open Source y es la base de multitud de creaciones muy importantes hoy en día.

Está diseñado para crear SPA (Single Page Application). En este tipo de aplicaciones web, todo lo que se muestra y se procesa está dentro de la misma página, así que al pasar de una opción a otra no hace falta recargar el navegador y lo normal es que trate de un único archivo desde el que se reproduce absolutamente todo.

Utiliza programación orientada a componentes. Estos componentes se representan como clases que heredan de la clase **Component**.

- Otros detalles:

No se suele considerar un framework.

Se utiliza para crear interfaces en el front-end.

Es la capa vista en un Modelo Vista Controlador

- Conceptos importantes:
 - Modelo Vista Controlador
 - Hay que recordar cómo funciona un Modelo Vista Controlador:
 1. El cliente (usuario) hace una petición al servidor.
 2. El controlador procesa la petición que hace el usuario y solicita los datos al modelo.
 3. El modelo se comunica con la base de datos y retorna la información.
 4. El modelo retorna los datos al controlador.
 5. El controlador retorna la vista al cliente con los datos solicitados.
 - Front-end y Back-end
 - El front-end abarca todo aquello que se refiere a la interfaz de usuario y la usabilidad de la aplicación.
 - El back-end procesa las interacciones del usuario (que le pasa el front-end) y realiza todos los procesos con los datos recibidos.

La documentación oficial está disponible en el siguiente [enlace](#).

1.1.- Lo necesario para utilizar React

Necesitamos tener conocimientos sobre las siguientes tecnologías:

- HTML y CSS
- JavaScript
- DOM: Document Object Model – Define la estructura básica de los elementos en documentos HTML y permite realizar modificaciones en tiempo real mediante la utilización de lenguajes como JavaScript).
- ES6, también llamada ECMAScript 2015 (especificación lanzada en 2015 que define estándares para JavaScript).
- Node.js y npm instalados de forma global.

1.2.- Preparación del entorno

Instalación de Node.js y npm de forma global (disponible para todos los proyectos, no para únicamente de forma local para un proyecto)

1.2.1.- ¿Qué es Node.js?

Node.js es un entorno en tiempo de ejecución que utiliza código abierto de JavaScript y que ha sido creado para generar aplicaciones web de forma muy optimizada.

En este caso cambia el concepto que tenemos de JavaScript, que normalmente lo hemos usado únicamente en la parte de cliente.

Mediante Node.js se proporciona un entorno de ejecución del lado del servidor que compila y ejecuta aplicaciones a velocidades muy altas. Utiliza un modelo asíncrono y dirigido por eventos.

1.2.2.- ¿Qué es npm?

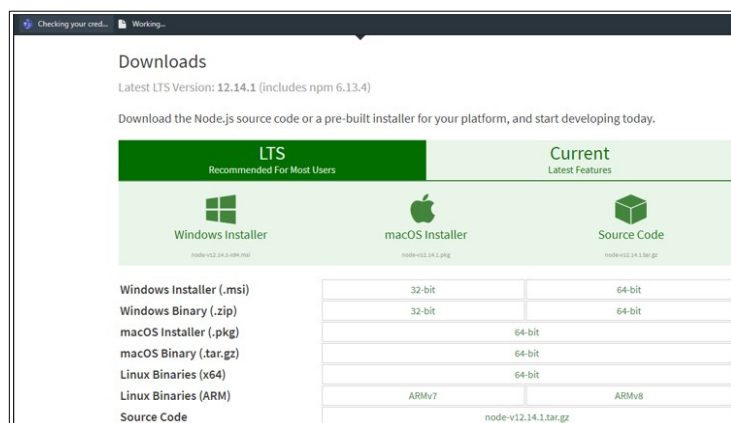
Es el sistema de gestión de paquetes que utiliza por defecto Node.js. Gracias a él podremos tener cualquier librería disponible con sólo una línea de código. Npm nos ayudará a administrar módulos, distribuir paquetes y agregar dependencias de una manera sencilla.

1.2.3.- Instalación de Node.js

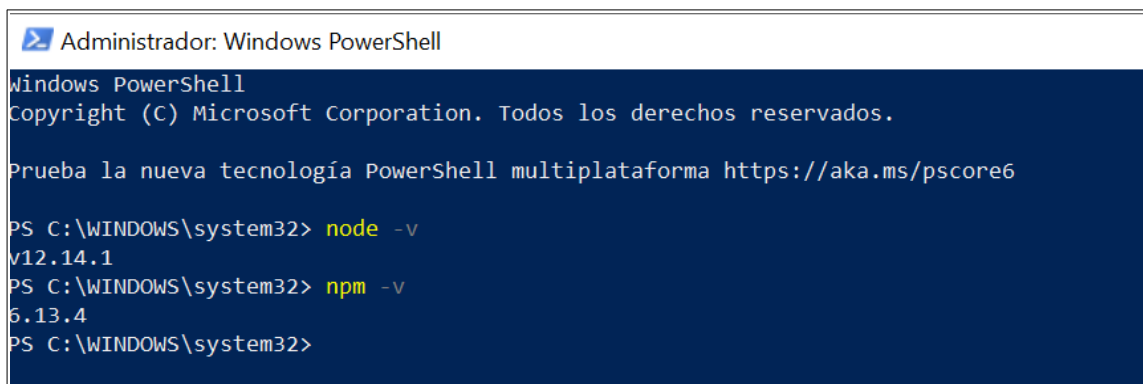
En este caso vamos a hacer la instalación en Windows, pero también podríamos implantarlo en Linux o Mac OS X porque se trata de software multiplataforma.

Debemos realizar la descarga desde la [web oficial](#). Nos interesa la versión LTS, que es mucho más estable.

Los pasos de instalación son muy sencillos:



Dejaremos las opciones de instalación por defecto, mediante PowerShell, podemos comprobar que **Node.js** y **npm** está correctamente instalados:



```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\WINDOWS\system32> node -v
v12.14.1
PS C:\WINDOWS\system32> npm -v
6.13.4
PS C:\WINDOWS\system32>
```

1.3.- Primeros pasos en React

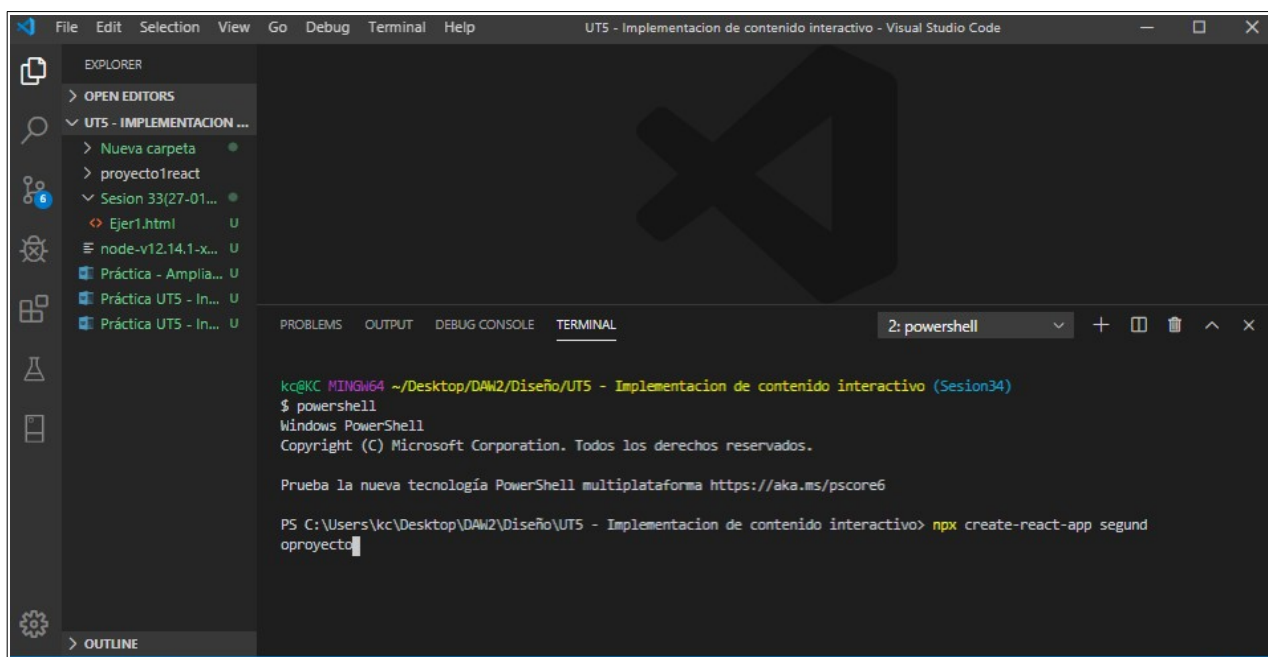
Creación de una aplicación React

Es posible realizar los preparativos de React mediante la utilización de un fichero estático html llamado “index.html” y la carga de varias librerías escritas en JavaScript, pero vamos a hacerlo de otra forma más directa y sencilla.

Para evitar lo antes nombrado, que puede ser bastante engorroso e ineficiente, Facebook ha creado un entorno preconfigurado llamado “**Create React App**”. Tenemos mucha documentación oficial [aquí](#). En nuestro caso lo vamos a hacer lo siguiente.

Para preparar “create-react-app” tenemos que ejecutar las siguientes órdenes en la terminal (cuidado, **React no permite mayúsculas en el nombre de los proyectos**). Vamos a proceder a la creación de nuestro primer proyecto mediante mediante “**create-react-app**”

```
npx create-react-app primerproyectorreact
```



```
File Edit Selection View Go Debug Terminal Help
UT5 - Implementacion de contenido interactivo - Visual Studio Code

EXPLORER
> OPEN EDITORS
> UT5 - IMPLEMENTACION ...
  > Nueva carpeta
  > proyecto1react
  > Sesión 33(27-01...
    < Ejer1.html
    < node-v12.14.1-x...
    < Práctica - Amplia...
    < Práctica UT5 - In...
    < Práctica UT5 - In...

TERMINAL
2: powershell
kc@KC MINGW64 ~/Desktop/DAW2/Diseño/UT5 - Implementacion de contenido interactivo (Sesion34)
$ powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\kc\Desktop\DAW2\Diseño\UT5 - Implementacion de contenido interactivo> npx create-react-app segund
oproyecto
```

Después de la preparación de “create-react-app”, estamos listos para cambiar al directorio del proyecto e iniciarlo mediante la orden “npm start”:

```
PS C:\Users\tiwx\OneDrive - Educacyl\IES María Moliner 2019-2020\Materiales alumnos\DIW\NuevoTemarioDesarrollado\ProyectoReact> cd .\primerproyectoreact\
PS C:\Users\tiwx\OneDrive - Educacyl\IES María Moliner 2019-2020\Materiales alumnos\DIW\NuevoTemarioDesarrollado\ProyectoReact\primerproyectoreact> npm start
> primerproyectoreact@0.1.0 start C:\Users\tiwx\OneDrive - Educacyl\IES María Moliner 2019-2020\Materiales alumnos\DIW\NuevoTemarioDesarrollado\ProyectoReact\primerproyectoreact
> react-scripts start
```

Una vez que hemos lanzado esta orden, se abrirá una pestaña en el navegador con el proyecto que acabamos de crear. Una peculiaridad respecto al desarrollo web al que estamos acostumbrados, es que no es necesario recargar la página ante los cambios, todo esto debido a que el servidor está en modo escucha ante posibles modificaciones y compilará el código después de una orden de guardado.

En la siguiente imagen podemos ver la página principal del proyecto (podemos modificar el código html para visualizar los cambios):

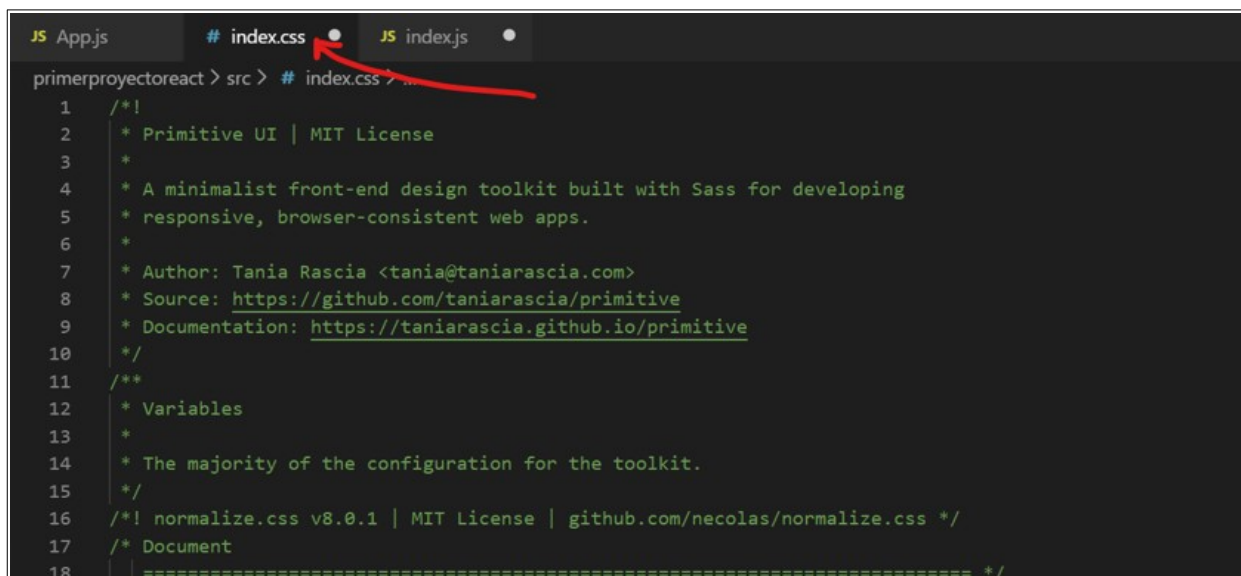
Si miramos la estructura de directorios del proyecto, vemos que es más compleja de lo normal, pero por el momento sólo nos vamos a fijar en el directorio src que contiene todo nuestro código React.

Vamos a borrar todos los ficheros del directorio /src, salvo:

- **index.js**
- **index.css**

V2 > Diseño > UT5 - Implementacion de contenido interactivo > proyecto1react > src					Buscar en ...
Nombre	Fecha de modificación	Tipo	Tamaño		
App.css	26/10/1985 10:15	Archivo CSS	1 KB		
App.js	26/10/1985 10:15	Archivo JavaScript	1 KB		
App.test.js	26/10/1985 10:15	Archivo JavaScript	1 KB		
index.css	26/10/1985 10:15	Archivo CSS	1 KB		
index.js	26/10/1985 10:15	Archivo JavaScript	1 KB		
logo.svg	26/10/1985 10:15	Documento SVG	3 KB		
serviceWorker.js	26/10/1985 10:15	Archivo JavaScript	5 KB		
setupTests.js	26/10/1985 10:15	Archivo JavaScript	1 KB		

En index.css vamos a copiar el código del framework de código abierto “Primitive CSS” para utilizar unas clases predefinidas y ahorrar trabajo, nada más. Podríamos personalizar perfectamente nuestro código CSS.



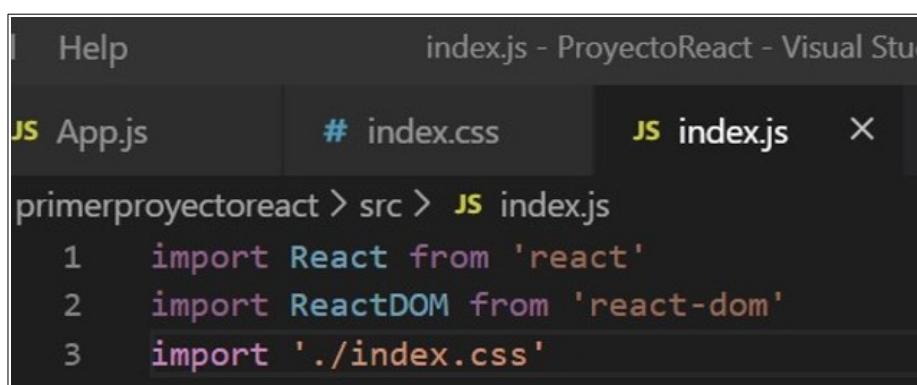
```

1  /*!
2  * Primitive UI | MIT License
3  *
4  * A minimalist front-end design toolkit built with Sass for developing
5  * responsive, browser-consistent web apps.
6  *
7  * Author: Tania Rascia <tania@taniarascia.com>
8  * Source: https://github.com/taniarascia/primitive
9  * Documentation: https://taniarascia.github.io/primitive
10 *
11 /**
12 * Variables
13 *
14 * The majority of the configuration for the toolkit.
15 */
16 /*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css */
17 /* Document
18 | ===== */

```

Cuidado, para empezar a crear el primer proyecto es mejor utilizar la estructura que se plantea en el siguiente apartado “Componentes en React”.

Ahora, en index.js debemos importar React, ReactDOM y el fichero CSS que acabamos de crear (no son necesarios los ; en esta especificación).



```

1  import React from 'react'
2  import ReactDOM from 'react-dom'
3  import './index.css'

```

A continuación, vamos a crear el componente App desde 0.

- App va a heredar de Component.
- Nos fijamos en que el div no va a usar “class”. En su lugar utiliza “classname” porque realmente no estamos escribiendo código HTML, estamos escribiendo código JavaScript.

Ahora indicamos que este componente App va a ser la raíz del proyecto:

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

Lanzamos el proyecto con el comando: `npm start`

Podemos visualizar el resultado en el navegador:

1.4.- Formas de escribir código en React

Es posible escribir código en React de varias formas, pero la más recomendable es utilizar JSX (JavaScript + XML).

JSX no es HTML, está más cerca de JavaScript.

Debemos tener en cuenta varios aspectos importantes:

- Se utiliza `className` en vez de `class` para aplicar clases CSS (esto es porque `class` es una palabra reservada en JavaScript).
- Las propiedades y métodos se aplicarán con la segunda parte del nombre en mayúscula (camelCase). Por ejemplo: `onclick` se convertirá en `onClick`
- Las etiquetas que se cierran en sí mismas, por ejemplo `img`, lo harán de esta forma: ``

1.5.- Variables en React

Declarar y asignar valores a variables en React utilizando JSX es muy sencillo:

```
const nombre = 'Kosta'
```

```
const encabezado = <h1>Hola, {nombre}</h1>
```

1.6.- Componentes en React

En React prácticamente cualquier cosa es un componente, que puede ser:

- Componentes de clase
- Componentes simples

Estos componentes los explicaremos por separado.

Las aplicaciones pueden estar compuestas de muchos pequeños componentes, de hecho, podríamos cambiar la estructura de la aplicación que acabamos de crear:

En el fichero **index.js** tendríamos:

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'
import './index.css'
ReactDOM.render(<App />, document.getElementById('root'))
```

Es decir, vamos a importar un componente `App` desarrollado en otro fichero `App.js`

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

class App extends Component {
  render() {
    return (
      <div className="App">
```



```
    <h1>¡Funciona!</h1>
  </div>
)
}
}
export default App
```

Es fundamental no olvidar que en este fichero debe ir la línea que nos permite exportar el componente “export default App”.

Puede ser interesante separar la aplicación en componentes, aunque no es necesario.

Es fundamental no olvidar que en este fichero debe ir la línea que nos permite exportar el componente “export default App”.

Puede ser interesante separar la aplicación en componentes, aunque no es necesario.

1.7.- Componentes de clase

Estos componentes serán desarrollados en componentes personalizados de clase.

Ya nos habremos fijado en que los componentes que hemos trabajado hasta la fecha están capitalizados, por ejemplo “**App**”.

Es fundamental capitalizar el nombre de los componentes para diferenciarlos de los elementos HTML.

En el siguiente ejemplo vamos a desarrollar un componente Tabla que vamos a rellenar con datos:

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

class Tabla extends Component {
  render() {
    return (
      <table>
        <thead>
          <tr>
            <th>Nombre</th>
            <th>Apellidos</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>Joel</td>
            <td>Edgerton</td>
          </tr>
          <tr>
            <td>Carmen</td>
            <td>Maura</td>
          </tr>
          <tr>
            <td>Luis</td>
            <td>Tosar</td>
          </tr>
          <tr>
            <td>Chloe</td>
          </tr>
        </tbody>
      </table>
    )
  }
}
```

```
        <td>Grace Moretz</td>
      </tr>
    </tbody>
  </table>
)
}
}
export default Tabla
```

Y en el fichero **App.js**

```
import React, { Component } from 'react'
import Tabla from './Tabla'

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>¡Funciona!</h1>
        <Tabla />
      </div>
    )
  }
}

export default App
```



<https://www.taniarascia.com/getting-started-with-react/>

1.8.- Componentes simples

Los componentes simples son básicamente funciones. No utilizan la palabra clave class.

Tomando como referencia el ejemplo anterior, vamos a crear dos componentes simples para formar una tabla:

- Una cabecera para la tabla
- Un body para la tabla

Vamos a utilizar la sintaxis de ES6 para crear estas funciones (funciones flecha, en este caso):

Esto lo vamos a implementar en un fichero **TablaComponentesSimples.js**

Primero el componente simple **TablaCabecera**

Posteriormente el componente simple **TablaCuerpo**

```
import React, { Component } from "react";

const TablaCabecera = () => {
  return (
    <thead>
      <tr>
        <th>Nombre:</th>
        <th>Apellidos:</th>
      </tr>
    </thead>
  )
}

const TablaCuerpo = () => {
  return (
    <tbody>
      <tr>
        <td>Joel</td>
        <td>Edgerton</td>
      </tr>
      <tr>
        <td>Carmen</td>
        <td>Maura</td>
      </tr>
      <tr>
        <td>Chloe</td>
        <td>Grace Moretz</td>
      </tr>
    </tbody>
  )
}
```

```
class TablaComponentesSimples extends Component{
  render(){
    return(
      <table>
        <TablaCabecera/>
        <TablaCuerpo/>
      </table>
    )
  }
}

export default TablaComponentesSimples
```

Hay que recordar que para poder utilizar este componente de clase debemos incluir la línea para exportarlo.

Cuando creamos componentes, nunca hay que olvidar esta instrucción:

Para componentes simples y componentes de clase:

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

Para componentes simples:

```
import React from 'react'
```

1.9.- Props

Son argumentos que se pasan a los componentes en React.

Podríamos decir que **Props** son argumentos que se pasan a los componentes en React. Sería algo similar a los parámetros que pasamos a una función en JavaScript.

Una característica de **Props** es que únicamente son de lectura, es decir, los pasamos como parámetros al componente, pero este componente no puede modificarlos. Si queremos modificar estos datos, tenemos que usar **State**.

Siguiendo el ejemplo de componente con cabecera y cuerpo, en este caso vamos a borrar todos los datos de esta tabla y vamos a crear una colección de datos (array de objetos) basada en JSON:

En **App.js**, concretamente dentro de **render()**, antes de **return**, por supuesto.

```
render(){
  /*Coleccion de objetos con nomenclatura JSON para utilizar con props */
  const actoresActrices = [
    {
      nombre: 'Joel',
      apellidos:'Edgerton'
    },
    {
      nombre: 'Carmen',
```

```
    apellidos: 'Maura'
  },
  {
    nombre: 'Luis',
    apellidos: 'Tosar'
  },
  {
    nombre: 'Chloe',
    apellidos: 'Grace Moretz'
  }
]
```

En **App.js** tenemos que pasar los datos a nuestro componente:

```
<TablaComponentesSimplesProps datosActoresActrices={actoresActrices}/>

return (
  <div className="App">
    <h2>Tabla creada mediante componente de clase que no
      utiliza componentes simples
    </h2>
    <Tabla />
    <h2>Tabla creada mediante un componente de clase que
      utiliza componentes simples
    </h2>
    <TablaComponentesSimples/>
    <h2>Tabla creada mediante un componente de clase que utiliza paso de parámetros con props</h2>
    /* Sesión 34 -> 30 de Enero de 2020 */
    <TablaComponentesSimplesProps datosActoresActrices={actoresActrices}/>
    <TablaComponentesSimplesState datosPersonaje={personajes} borrarPersonaje={this.borrarPersonaje}/>
    <h2>Añadir nuevo personaje</h2>
    <Formulario manejarEnvio={this.manejarEnvio} />
  </div>
)
```

Como hemos pasado los datos a ese componente, tenemos que modificar la parte del propio componente en **TablaComponentesSimplesProps.js**

```
class TablaCamponentesSimplesProps extends Component {
  render() {
    const { datosActoresActrices } = this.props
    return (
      <table>
        <TablaCabecera />
        <TablaCuerpo datosActoresActrices={datosActoresActrices} />
      </table>
    )
  }
}
```

Es importante saber que podemos acceder a todos los props desde un componente mediante **this.props**. En nuestro caso sólo estamos pasando un props, pero para acceder a uno en concreto deberíamos utilizar **props.nombreprops**

Seguimos en **TablaComponentesSimplesProps.js**.

Como podemos observar a continuación, utilizaremos el props como parámetro para el componente simple **TablaCuerpo**. Utilizamos el método map para retornar una fila de tabla por cada objeto del array **datosActoresActrices**. Con índice llevamos la cuenta del número de filas. Hay que tener en cuenta que vamos a utilizar una key para identificar el elemento dentro del array. Las keys sólo se suelen utilizar cuando el orden de estos elementos no va a cambiar.

Todos estos datos los vamos a recoger en una variable **filasDeDatos** que vamos a utilizar en el componente simple.

```
const TablaCuerpo = props => {
  const filasDeDatos = props.datosActoresActrices.map((fila, indice) => {
    return (
      <tr key={indice}>
        <td>{fila.nombre}</td>
        <td>{fila.apellidos}</td>
      </tr>
    )
  })
  return (
    <tbody>
      {filasDeDatos}
    </tbody>
  )
}
```

Seguimos en **TablaComponentesSimplesProps.js**

Nuestro componente de clase, en su parte cargada mediante el componente simple **TablaCuerpo**, va a utilizar el props que hemos pasado desde **App.js**

```
class TablaCamponentesSimplesProps extends Component {
  render() {
    const { datosActoresActrices } = this.props
    return (
      <table>
```

```
        <TablaCabecera />
        <TablaCuerpo datosActoresActrices={datosActoresActrices} />
    </table>
  )
}
}

export default TablaCamponentesSimplesProps
```

Codigo completo de los ejercicios realizados en clase:

```
import React,{ Component } from "react";

/*Sacado de TablaComponentesSimple.js */
const TablaCabecera = () => {
  return (
    <thead>
      <tr>
        <th>Nombre:</th>
        <th>Apellidos:</th>
      </tr>
    </thead>
  )
}

/*En este caso en el cuerpo de la tabla pasaremos parametros, asi se podra const
ruir la tabla con los datos
pasados */
const TablaCuerpo = props => {
  const filasDeDatos = props.datosActoresActrices.map((fila, indice) => {
    return (
      <tr key={indice}>
        <td>{fila.nombre}</td>
        <td>{fila.apellidos}</td>
      </tr>
    )
  })

  return (
    <tbody>
      {filasDeDatos}
    </tbody>
  )
}
```



```
}  
  
class TablaComponentesSimplesProps extends Component {  
  render() {  
    /* <TablaComponentesSimplesProps datosActoresActrices={actoresActrices}  
    /> */  
    const { datosActoresActrices } = this.props /*Utilizo props pasaos al com  
    ponente desde la aplicacion*/  
  
    return (  
      <table>  
        <TablaCabecera />  
        <TablaCuerpo datosActoresActrices={datosActoresActrices} />  
      </table>  
    )  
  }  
}  
  
export default TablaComponentesSimplesProps
```

Es fundamental saber que el componente no puede cambiar los valores del props.

1.10.- State – Permite modificar los datos de un componente

Props tiene una utilidad muy importante, sin embargo no nos permite cambiar los datos de un componente. Para esto utilizamos **state**.

Podemos pensar en **State** como datos que pueden ser guardados y modificados sin necesidad de ser añadidos a una base de datos.

En App.js antes de render()

Vamos a crear un objeto State con los nombres de unos personajes de series.

```
/*Objeto para utilizar en state*/
state = {
  personajes: [
    {
      name: 'Payton Hobart',
    },
    {
      name: 'Wendy Carr',
    },
    {
      name: 'Mina',
    },
    {
      name: 'Jonathan Harker',
    },
    {
      name: 'Drácula',
    },
    {
      name: 'Once',
    },
    {
      name: 'Jim Hopper',
    }
  ]
}
```

En App.js antes de render()

También vamos a implementar un método para borrar personajes:

```
/*Método para borrar personajes*/
borrarPersonaje = indice => {
  const { personajes } = this.state;

  this.setState({
    personajes: personajes.filter((personaje, i) => {
      return i !== indice;
    })
  });
}
```

Inicializamos el objeto personajes, ahora dentro de **render()** en App.js

```
/*Para utilizar en state*/
const { personajes } = this.state
```

Props tiene una utilidad muy importante, sin embargo no nos permite cambiar los datos de un componente. Para esto utilizamos state.

Podemos pensar en State como datos que pueden ser guardados y modificados sin necesidad de ser añadidos a una base de datos.

En App.js antes de render()

Vamos a crear un objeto State con los nombres de unos personajes de series.

```
/*Objeto para utilizar en state*/

state = {
  personajes: [
    {
      name: 'Payton Hobart',
    },
    {
      name: 'Wendy Carr',
    },
    {
```

```
    name: 'Mina',
  },
  {
    name: 'Jonathan Harker',

  },
  {
    name: 'Drácula',
  },
  {
    name: 'Once',
  },
  {
    name: 'Jim Hopper',
  }
]
}
```

En App.js antes de render()

También vamos a implementar un método para borrar personajes:

```
/*Método para borrar personajes*/
borrarPersonaje = indice => {
  const { personajes } = this.state;

  this.setState({
    personajes: personajes.filter((personaje, i) => {
      return i !== indice;
    })
  });
};
}
```

Inicializamos el objeto personajes, ahora dentro de **render()** en **App.js**

```
/*Para utilizar en state*/  
const { personajes } = this.state
```

Ahora, dentro de **return ()**, en **App.js**, pintamos el componente simple (para trabajar con state es conveniente convertir el componente de clase en uno simple):

```
<h1>Tabla creada con un componente simple (con state y que permite eliminar elementos) que utiliza dos componentes simples</h1>  
    <TablaComponentesSimplesState datosPersonaje={personajes} borrarPersonaje={this.borrarPersonaje} />
```

Ahora debemos crear un componente simple **TablaComponentesSimplesStat.js**

El inicio es idéntico a los otros componentes de Tabla que hemos creado:

```
import React from 'react'  
  
const TablaCabecera = () => {  
  return (  
    <thead>  
      <tr>  
        <th>Nombre</th>  
        <th>Apellidos</th>  
      </tr>  
    </thead>  
  )  
}
```

Ahora debemos modificar el componente simple TablaCuerpo con el acceso a los datos state:

```
const TablaCuerpo = props => {
  const filasDeDatos = props.datosPersonaje.map((fila, indice) => {
    return (
      <tr key={indice}>
        <td>{fila.name}</td>
        <td><button onClick={() => props.borrarPersonaje(indice)}>Borrar
</button></td>
      </tr>
    )
  })
  return (
    <tbody>
      {filasDeDatos}
    </tbody>
  )
}
```

Convertimos el componente de clase en un componente simple para poder utilizar los datos state:

```
const TablaComponentesSimplesState = (props) => {

  const { datosPersonaje, borrarPersonaje } = props;
  return (
    <table>
      <TablaCabecera />
      <TablaCuerpo datosPersonaje={datosPersonaje} borrarPersonaje={bo
rrarPersonaje} />
    </table>
  )
}
```

```
export default TablaComponentesSimpleState
```

Ahora ya tenemos nuestro componente funcional en el que podemos eliminar datos: