

Unidad 2: TUTORIAL CREACION DE COMPONENTES CON REACT

## Introducción a Node y npm

### ¿Qué es Node.js?

*Node.js* es un entorno de ejecución de JavaScript que nos permite ejecutar código JavaScript fuera del navegador. Se puede ejecutar en cualquier plataforma, es un proyecto en código abierto disponible para ejecutar en Windows, Linux, Unix, macOS y más. Esto significa que podemos crear aplicaciones de servidor, herramientas de línea de comandos y mucho más, todo utilizando el lenguaje que ya conocemos: JavaScript.

### ¿Qué es npm?

npm (Node Package Manager) es el gestor de paquetes oficial de Node.js y viene incluido en la instalación. Nos permite instalar, actualizar y eliminar paquetes (módulos de código reutilizable) de forma sencilla. Estos paquetes pueden ser desde pequeñas utilidades hasta frameworks completos como Express. En <https://www.npmjs.com/> puedes tener más información de los paquetes y sus versiones.

### Instalación de Node.js y npm

1. **Descarga:** Visita la página oficial de Node.js ([https://nodejs.org/](https://www.google.com/url?sa=E&source=gmail&q=https://nodejs.org/)) y descarga el instalador adecuado para tu sistema operativo.
2. **Instalación:** Sigue las instrucciones del instalador para completar la instalación. Normalmente, esto incluye agregar Node.js y npm a tu PATH, lo que te permitirá ejecutarlos desde cualquier directorio de tu terminal.

### Verificando la instalación

Una vez instalada, puedes verificar la versión de Node.js y npm ejecutando los siguientes comandos en tu terminal:

>> node -v

>> npm -v

### Creando tu primer proyecto Node.js

**Crear un directorio:** Abre tu terminal y crea un nuevo directorio para tu proyecto:

>> mkdir mi-primer-proyecto

>> cd mi-primer-proyecto

**Inicializar npm:** Inicializa un proyecto npm ejecutando el siguiente comando:

npm init -y

Esto creará un archivo package.json que contiene información sobre tu proyecto, como las dependencias.

**Ejercicio: "Hola, Mundo!" en Node.js**

1. **Crear un archivo:** Crea un archivo llamado index.js en tu proyecto.
2. **Escribir el código:** Abre el archivo index.js y escribe el siguiente código:

console.log("¡Hola, mundo desde Node.js!");

1. **Ejecutar el código:** En tu terminal, ejecuta el siguiente comando:

node index.js

## Introducción a Bootstrap

Bootstrap fue una herramienta originalmente desarrollada por ingenieros en Twitter: Mark Otto y Jacob Thorton como un intento de fomentar la utilización del mismo framework para minimizar las inconsistencias en el equipo de ingeniería de Twitter. Ni que decir tiene, que la iniciativa de Bootstrap triunfó en Twitter puesto que el equipo completo comentó a trabajar más rápido, de forma más eficaz y con menos inconsistencias.

Aunque comenzó como una solución interna en Twitter, Mark y Jacob, pronto se dieron cuenta de que aquello tendría un gran potencial. En Agosto de 2011, [el framework Bootstrap fue lanzado al público](https://blog.twitter.com/2011/bootstrap-twitter) como proyecto Open Source en Github. En los siguientes meses, miles de desarrolladores de todo el mundo contribuyeron al proyecto y Bootstrap se convirtió en el proyecto Open Source más activo del mundo. Desde entonces, Bootstrap ha ganado más notoriedad y se ha convertido en “el framework de presentación más popular para desarrollar proyectos responsive y para móviles en Internet”.

Bootstrap es una colección de varios elementos web personalizables y funciones completamente empaquetado en una sola herramienta. Cuando se diseña una web con Bootstrap, los desarrolladores pueden elegir qué elementos utilizar. Aún más importante, tienen la certeza de saber que los elementos que elijan no generarán conflictos entre ellos. Como si se tratase de un puzle, exceptuando que cada pieza del puzle encaja perfectamente con las otras, sin importar la pieza que elija.

Entre sus ventajas:

* **Rapidez de desarrollo:** Al proporcionar componentes y estilos predefinidos, Bootstrap acelera significativamente el proceso de desarrollo web.
* **Consistencia:** Asegura una apariencia y comportamiento uniformes en toda tu aplicación.
* **Responsividad:** Permite crear diseños que se adaptan automáticamente a diferentes tamaños de pantalla.
* **Gran comunidad:** Cuenta con una comunidad activa y una amplia documentación, lo que facilita encontrar soluciones y aprender.

### Instalación de Bootstrap

Existen varias formas de incluir Bootstrap en tu proyecto:

* **CDN (Content Delivery Network):** La forma más rápida es incluir los archivos CSS y JavaScript de Bootstrap directamente desde un CDN en la cabecera de tu HTML.

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>Bootstrap demo</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN" crossorigin="anonymous">

<link rel="stylesheet" href="styles.css">

</head>

* **Descarga:** Puedes descargar los archivos y alojarlos en tu propio servidor.
* **npm:** Si estás utilizando un gestor de paquetes como npm, puedes instalar Bootstrap como una dependencia de tu proyecto.

Crearemos un Proyecto node al que le añadimos la dependencia

mkdir my-bootstrap-project

cd my-bootstrap-project

npm init -y

npm install bootstrap

En el fichero inicial (index.html) añadiremos las cabeceras:

<!DOCTYPE html>

<html>

<head>

<title>My Bootstrap Project</title>

<link rel="stylesheet" href="node\_modules/bootstrap/dist/css/bootstrap.min.css">

</head>

<body>

<script src="node\_modules/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<div class="container-fluid p-5 bg-primary text-white text-center">

<h1>My First Bootstrap Page</h1>

<p>Resize this responsive page to see the effect!</p>

</div>

<div class="container mt-5">

<div class="row">

<div class="col-sm-4">

<h3>Column 1</h3>

<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>

<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>

</div>

<div class="col-sm-4">

<h3>Column 2</h3>

<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>

<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>

</div>

<div class="col-sm-4">

<h3>Column 3</h3>

<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>

<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>

</div>

</div>

</div>

</body>

</html>

### Sistema de cuadrículas

Bootstrap se basa en un sistema de cuadrículas , en el cual vamos a incluir nuestros componentes a través de clases y atendiendo a cada dispositivo. Es decir, sabiendo que la página se va a mostrar como una cuadrícula en la que cada fila está compuesta de 12 cuadros, estableceremos para cada elemento de nuestra página cuantos cuadros va a ocupar si estamos en un dispositivo muy pequeño (móvil), en un dispositivo pequeño (tablets), dispositivos de pantalla media (ordenadores), dispositivos de pantalla grande (grandes pantallas). Para ello utilizaremos elementos de clase .col-xs, .col-sm, .col-md, col-lg respectivamente y un sufijo como número del 1 al 12 , que indica el número de cuadrados.

* .col- (extra small devices – pantallas menores a 576px)
* .col-sm- (small devices – pantallas mayores o igual a 576px)
* .col-md- (medium devices - pantallas mayores o igual a 768px)
* .col-lg- (large devices - pantallas mayores o igual a 992px)
* .col-xl- (xlarge devices - pantallas mayores o igual a 1200px)
* .col-xxl- (xxlarge devices - pantallas mayores o igual a 1400px)

#### Ejemplo 1. Linea horizontal con celdas mismo tamaño para un solo dispositivo

<!DOCTYPE html>

<html>

<head>

<title>My Bootstrap Project</title>

<link rel="stylesheet" href="node\_modules/bootstrap/dist/css/bootstrap.min.css">

<style type="text/css">

.col-md-1, .col-md-2, .col-md-3, .col-md-4, .col-md-6, .col-md-8 {

background-color: #34495e;

color: white;

line-height: 5em;

border: 1px solid white;

height: 5em;

}

</style>

</head>

<body>

<script src="node\_modules/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

<div class="container-fluid p-5 bg-primary text-white text-center">

<h1>My First Bootstrap Page</h1>

<p>Resize this responsive page to see the effect!</p>

</div>

<div class="container">

<div class="row">

<div class="col-md-1">1</div>

<div class="col-md-1">2</div>

<div class="col-md-1">3</div>

<div class="col-md-1">4</div>

<div class="col-md-1">5</div>

<div class="col-md-1">6</div>

<div class="col-md-1">7</div>

<div class="col-md-1">8</div>

<div class="col-md-1">9</div>

<div class="col-md-1">10</div>

<div class="col-md-1">11</div>

<div class="col-md-1">12</div>

</div>

</body>

</html>

#### Ejemplo 2. celdas distinto tamaño para un solo dispositivo

style type="text/css">

.col-md-1, .col-md-2, .col-md-3, .col-md-4, .col-md-6, .col-md-8 {

background-color: #34495e;

color: white;

line-height: 5em;

border: 1px solid white;

height: 5em;

}

</style>

……

<div class="container">

<div class="row">

<div class="col-md-4">1.1</div>

<div class="col-md-4">1.2</div>

<div class="col-md-4">1.3</div>

</div>

<div class="row">

<div class="col-md-3">2.1</div>

<div class="col-md-3">2.2</div>

<div class="col-md-3">2.2</div>

<div class="col-md-3">2.3</div>

</div >

<div class="row">

<div class="col-md-6">3.1</div>

<div class="col-md-6">3.2</div>

</div >

<div class="row">

<div class="col-md-8">4.1</div>

<div class="col-md-4">4.2</div>

</div >

#### Ejemplo 3. celdas distinto tamaño según dispositivo

<div class="container">

<h1>Medium /Small Device</h1>

<p> </p>

<p>Resize the browser window to see the effect.</p>

<div class="row">

<div class="col-sm-3 col-md-6" style="background-color:yellow;color: darkolivegreen; ">

Esto es el contenido de la celda 1

</div>

<div class="col-sm-9 col-md-6" style="background-color:pink; color: #34495e;">

Esto es el contenido de la celda 2

</div>

</div>

</div>

**Offsetting columns**

La opción de Offsetting nos va a permitir indicar que x número de cuadrados de la cuadrícula

deben quedar vacíos entre columnas definidas, para ello introducimos col-\*-offset.

#### Ejemplo 4. Inserción Offsets :

<div class="container">

<div class="row">

<div class="col-md-4">.col-md-4</div >

<div class="col-md-4 offset-md-4">.col-md-4 .offset-md-4</div >

</div >

<div class="row">

<div class="col-md-3 offset-md-3">.col-md-3 .offset-md-3"</div >

<div class="col-md-3 offset-md-3">.col-md-3 .offset-md-3"</div >

</div >

<div class="row">

<div class="col-md-6 offset-md-3">.col-md-6 .offset-md-3</div ></div >

</div>

**Helpers - Elementos ocultos y visibles**

En muchas ocasiones necesitamos esconder algún elemento y mostrar otros dependiendo del

tamaño de pantalla. Para ello, bootstrap nos permite utilizar las clases NONE y BLOCK:

|  |  |
| --- | --- |
| Clase | Descripción |
| .d-none | Oculta el elemento en todos los tamaños de pantalla. |
| .d-block | Muestra el elemento en todos los tamaños de pantalla. |
| .d-sm-none | Oculta el elemento en pantallas pequeñas (576px o menos). |
| .d-sm-block | Muestra el elemento en pantallas pequeñas (576px o menos). |
| .d-md-none | Oculta el elemento en pantallas medianas (768px o menos). |
| .d-md-block | Muestra el elemento en pantallas medianas (768px o menos). |
| .d-lg-none | Oculta el elemento en pantallas grandes (992px o menos). |
| .d-lg-block | Muestra el elemento en pantallas grandes (992px o menos). |
| .d-xl-none | Oculta el elemento en pantallas extra grandes (1200px o menos). |
| .d-xl-block | Muestra el elemento en pantallas extra grandes (1200px o menos). |
| .d-xxl-none | Oculta el elemento en pantallas extra extra grandes (1400px o menos). |
| .d-xxl-block | Muestra el elemento en pantallas extra extra grandes (1400px o menos). |

**Ejemplo1.La pantalla siguiente esconde texto para distintos tamaños, de tal manera que si estamos es un dispositivos medio solo veremos mensajes de todas los demás dispositivos.**

<div class="container">

<h2>Example</h2>

<p>Resize this page to see how the text below changes:</p>

<h1 class="d-sm-none bg-info">This text is hidden on a SMALL screen.</h1>

<h1 class="d-md-none bg-warning">This is text hidden on a MEDIUM screen.</h1>

<h1 class="d-lg-none bg-success">This is text hidden on a LARGE screen.</h1>

<h1 class="d-xl-none bg-primary">This is text hidden on a extra LARGE screen.</h1>

<h1 class="d-xxl-none bg-info">This is text hidden on a extra extra LARGE screen.</h1>

</div>

Resultado:

<div class="container">

<h2>Example</h2>

<p>Resize this page to see how the text below changes:</p>

<h2>Example</h2>

<h1 class="d-block d-sm-none bg-info">This text is visible on a SMALL screen.</h1>

<h1 class="d-none d-sm-block d-md-none bg-warning">This is text visible on a MEDIUM screen.</h1>

<h1 class="d-none d-md-block d-xl-none bg-success">This is text visible on a LARGE screen.</h1>

<h1 class="d-none d-xl-block d-xxl-none bg-primary">This is text visible on a extra LARGE screen.</h1>

</div>

### Componentes Bootstrap 5: Construyendo bloques para tus proyectos

**Introducción a los componentes**

En esta unidad exploraremos en detalle los componentes de Bootstrap 5, que son elementos prediseñados que facilitan la creación de interfaces de usuario consistentes y responsivas. Cada componente ofrece una amplia gama de opciones de personalización para adaptarse a tus necesidades específicas.

**Componentes básicos**

* **Card:** Un contenedor flexible para presentar contenido, ideal para mostrar noticias, productos o cualquier tipo de información en una estructura organizada.
* **Button:** Botones con diferentes estilos y tamaños para realizar acciones.
* **Navbar:** Crea barras de navegación personalizadas para la navegación principal de tu sitio web.
* **Form:** Componentes para crear formularios de contacto, de registro, etc., con validación integrada.
* **Table:** Componentes para mostrar datos tabulares de forma organizada.

**Personalización de componentes**

* **Clases de utilidad:** Descubre cómo utilizar las clases de utilidad de Bootstrap para personalizar el espaciado, el tamaño, el color y otros aspectos de los componentes.
* **Variables CSS:** Aprende a modificar las variables CSS de Bootstrap para crear una apariencia completamente personalizada.
* **JavaScript:** Interactúa con los componentes utilizando JavaScript para agregar funcionalidades como modales, tooltips y carruseles.

**Componentes avanzados**

* **Modal:** Ventanas modales para mostrar contenido adicional o formularios.
* **Carousel:** Componentes para crear carruseles de imágenes o contenido.
* **Collapse:** Elementos que se pueden expandir y contraer.
* **Dropdown:** Menús desplegables para ofrecer opciones adicionales.
* **Pagination:** Componentes para paginar listas largas de elementos

Vamos a ver algunos de estos componentes más en profundidad, que luego ampliaremos con todos los componentes cuando trabajemos con React:

**Card**

Una tarjeta en Bootstrap 5 es un cuadro con bordes y algo de relleno alrededor de su contenido. Incluye opciones para encabezados, pies de página, contenido, colores, etc

<div class="container mt-3">

<h2>Card Image</h2>

<p>Image at the top (card-img-top):</p>

<div class="card" style="width:400px">

<img class="card-img-top" src="https://cdn.shopify.com/s/files/1/0509/9552/7861/files/Equipamiento\_gimnasio\_accesorios\_para\_las\_diferentes\_salas.jpg?v=1653463328" alt="Card image" style="width:100%">

<div class="card-body">

<h4 class="card-title">Máquinas</h4>

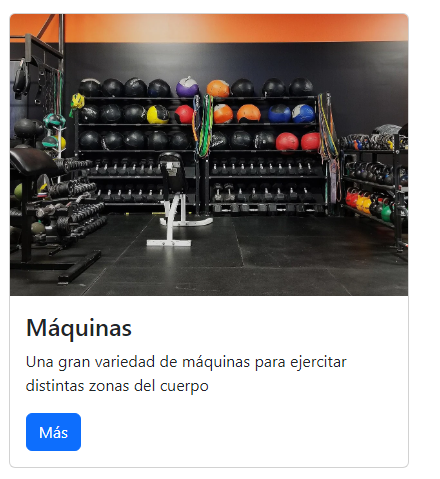
<p class="card-text">Una gran variedad de máquinas para ejercitar distintas zonas del cuerpo</p>

<a href="#" class="btn btn-primary">Más</a>

</div>

</div>

<br>



<button type="button" class="btn">Basic</button>  
<button type="button" class="btn btn-primary">Primary</button>  
<button type="button" class="btn btn-secondary">Secondary</button>  
<button type="button" class="btn btn-success">Success</button>  
<button type="button" class="btn btn-info">Info</button>  
<button type="button" class="btn btn-warning">Warning</button>  
<button type="button" class="btn btn-danger">Danger</button>  
<button type="button" class="btn btn-dark">Dark</button>  
<button type="button" class="btn btn-light">Light</button>  
<button type="button" class="btn btn-link">Link</button>



<nav class="navbar navbar-expand-sm bg-dark navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="#">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR7ZmgJjPgnK54V0R338xoNpOK-Lz6zh9pmz17OxKtmIP-PocCZ4OOe5Od4QEUKFlsxc3Y&usqp=CAU" alt="Cifp viles Logo" style="width:40px;" class="rounded-pill">

</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#collapsibleNavbar">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" >

<ul class="navbar-nav">

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class="dropdown-menu">

<li><a class="dropdown-item" href="#">FP Básica</a></li>

<li><a class="dropdown-item" href="#">FP Grado Medio</a></li>

<li><a class="dropdown-item" href="#">FP Grado Superior</a></li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js">Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="formacion.js">Formación</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js">Novedades</a>

</li>

</ul>

</div>

</div>

</nav>

Paso a paso :

1)Crear la lista desordenada con los elementos del menú

<ul>

<li> Formación

<ul>

<li>

FP Básica

</li>

<li>

FP Grado Medio

</li>

<li>

FP Grado Superior

</li>

</ul>

</li>

<li> Secretaría </li>

<li> Novedades </li>

</ul>

2)A todos los elementos del menú que aparecen en primer nivel se le añade un enlace con el menú a de la clase nav-link y href la URL donde nos direccionará

<ul>

<li> <a class="nav-link" href="formación.js" >Formación</a>

<ul>

<li>

FP Básica

</li>

<li>

FP Grado Medio

</li>

<li>

FP Grado Superior

</li>

</ul>

</li>

<li>

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li>

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

3) Añadir a los elementos de primer nivel la clase nav-item

<ul>

<li class="nav-item"> <a class="nav-link" href="formación.js" >Formación</a>

<ul>

<li>

FP Básica

</li>

<li>

FP Grado Medio

</li>

<li>

FP Grado Superior

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

4) Al elemento inicial de la lista principal le añadimos la clase "navbar-nav"

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link" href="formación.js" >Formación</a>

<ul>

<li>

FP Básica

</li>

<li>

FP Grado Medio

</li>

<li>

FP Grado Superior

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

5) Añadir a cada subelemento de la lista de segundo nivel un enlace con las siguientes clases

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link" href="formación.js" >Formación</a>

<ul>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Básica </a>

</li>

<li>

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li>

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

7) Vamos a indicar la lista de nivel 2 como "dropdown-menu" y modificar el enlace formación

<ul class="navbar-nav">

<li class="nav-item><a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class=="dropdown-menu">

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Básica </a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

8) Vamos a meter nuestra lista en un elemento div collapse

<div class="collapse navbar-collapse" >

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class="dropdown-menu">

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Básica </a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

</div>

9 ) Vamos a crear ya el menú como un menú de navegación introduciéndolo de estilo dark

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">

<div class="container-fluid">

<div class="collapse navbar-collapse" >

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class="dropdown-menu">

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Básica </a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

</div>

</div>

</nav>

10) Nos queda indicar el botón bocadillo que se mostrará cuando colapse(menú hamburguesa)

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">

<div class="container-fluid">

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#collapsibleNavbar">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id= collapsibleNavbar >

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class="dropdown-menu">

<li class="nav-item dropdown">

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Básica </a>

</li>

<li class="nav-item dropdown">

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li class="nav-item dropdown">

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

</div>

</div>

</nav>

11) Añadimos el logo de forma redondeada

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="#">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR7ZmgJjPgnK54V0R338xoNpOK-Lz6zh9pmz17OxKtmIP-PocCZ4OOe5Od4QEUKFlsxc3Y&usqp=CAU" alt="Cifp viles Logo" style="width:40px;" class="rounded-pill">

</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#collapsibleNavbar">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id= collapsibleNavbar >

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class="dropdown-menu">

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Básica </a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

</div>

</div>

</nav>

Podemos añadir al submenú el estilo data-bs-theme="dark"

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="#">

<img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcR7ZmgJjPgnK54V0R338xoNpOK-Lz6zh9pmz17OxKtmIP-PocCZ4OOe5Od4QEUKFlsxc3Y&usqp=CAU" alt="Cifp viles Logo" style="width:40px;" class="rounded-pill">

</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#collapsibleNavbar">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id= collapsibleNavbar >

<ul class="navbar-nav">

<li class="nav-item"> <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Formación</a>

<ul class="dropdown-menu" data-bs-theme="dark">

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Básica </a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown">FP Grado Medio</a>

</li>

<li >

<a class="dropdown-item" href="#" role="button" data-bs-toggle="dropdown"> FP Grado Superior</a>

</li>

</ul>

</li>

<li class="nav-item">

<a class="nav-link" href="secretaria.js" > Secretaría</a>

</li>

<li class="nav-item">

<a class="nav-link" href="novedades.js" >Novedades </a>

</li>

</ul>

</div>

</div>

</nav>

Componente Carousel. Solo lo subrayado en amarillo es lo que forma un carrousel. Los otros son elementos opcionales para añadir un indicador de imagen o los controles de derecha e izquierda o los textos

<!-- Carousel -->

<div id="demo" class="carousel slide" data-bs-ride="carousel">

<!-- Indicators/dots -->

<div class="carousel-indicators">

<button type="button" data-bs-target="#demo" data-bs-slide-to="0" class="active"></button>

<button type="button" data-bs-target="#demo" data-bs-slide-to="1"></button>

<button type="button" data-bs-target="#demo" data-bs-slide-to="2"></button>

</div>

<!—El carrousel -->

<div class="carousel-inner">

<div class="carousel-item active">

<img src="la.jpg" alt="Los Angeles" class="d-block" style="width:100%">

<div class="carousel-caption">

<h3>Los Angeles</h3>

<p>We had such a great time in LA!</p>

</div>

</div>

<div class="carousel-item">

<img src="chicago.jpg" alt="Chicago" class="d-block" style="width:100%">

<div class="carousel-caption">

<h3>Chicago</h3>

<p>Thank you, Chicago!</p>

</div>

</div>

<div class="carousel-item">

<img src="ny.jpg" alt="New York" class="d-block" style="width:100%">

<div class="carousel-caption">

<h3>New York</h3>

<p>We love the Big Apple!</p>

</div>

</div>

</div>

<!—Controles derecha e izda-->

**<button class="carousel-control-prev" type="button" data-bs-target="#demo" data-bs-slide="prev">**

**<span class="carousel-control-prev-icon"></span>**

**</button>**

**<button class="carousel-control-next" type="button" data-bs-target="#demo" data-bs-slide="next">**

**<span class="carousel-control-next-icon"></span>**

**</button>**

</div>

## Desarrollo con React

React es una biblioteca de JavaScript popular para crear interfaces de usuario de aplicaciones web, basado en componentes. La creación de una interfaz implicará el añadir cada uno de esos componentes , como en una casa de bloques de Lego. En React, cada bloque sería un componente, y al unirlos puedes construir la interfaz de usuario deseada.

### Componentes: Los Ladrillos de React

Un componente en React es como una pieza independiente de tu interfaz de usuario. Puede ser algo tan simple como un botón, un título o algo más complejo como un formulario completo. Lo mejor de todo es que puedes reutilizar estos componentes en diferentes partes de tu aplicación.

Ejemplo de componente

import React from 'react';

function Saludo(props) {

return <h1>¡Hola, {props.nombre}!</h1>;

}

export default Saludo;

En este ejemplo, *Saludo* es un componente que recibe un nombre como prop y muestra un mensaje de saludo personalizado.

Una vez que has construido tu componente, puedes anidar este componente en otros:

export default function MyApp() {

return (

<div>

<h1> Boletín de Noticias </h1>

<Saludo nombre="Irene"> </Saludo>

</div>

);

}

Observa que <Saludo> comienza con una letra mayúscula. Eso es cómo sabes que es un componente de React. Los nombres de los componentes de React siempre deben comenzar con una letra mayúscula, mientras que las etiquetas HTML deben estar en minúscula.

### Código declarativo con JSX

El código de retorno de nuestra componente se denomina JSX. (mezcla de HTML y código).

Aunque no es obligatorio es ampliamente usado

Ej: Código con JSX

const myElement = <h1>I Love JSX!</h1>;

Lo mismo sin JSX

const myElement = React.createElement('h1', {}, 'I Love JSX!');

JSX es más estricto que HTML, tienes que cerrar todas las etiquetas como <br/>. Un componente no puede devolver múltiples etiquetas raíz, solo una etiqueta raíz. Tienes que envolverlos en un padre compartido:

export default function MyApp() {

return (

<>

<h1> Boletín de Noticias </h1>

<Saludo nombre="Irene"> </Saludo>

</>

);

}

#### Añadiendo estilos

En React JSX se especifica una clase CSS con className

. Funciona de la misma manera que el atributo class de HTML:

export default function MyApp() {

return (

<>

<h1 className="bg-info"> Boletín de Noticias </h1>

<Saludo nombre="Irene"> </Saludo>

</>

);

}

#### Expresiones en JSX

Con JSX puedes escribir expresiones dentro de llaves {}. La expresión puede ser una variable de React, una propiedad o cualquier otra expresión válida de JavaScript. JSX ejecutará la expresión y devolverá el resultado:

import React from 'react';

function Calculadora() {

const numero1 = 5;

const numero2 = 3;

return (

<div>

<p>La suma de { numero1 } y {numero2 } es: {numero1 + numero2}</p>

</div>

);

}

Puedes incluir expresiones más complejas, como el siguiente caso en estilo, donde el estilo depende de variables

function Profile() {

const user = {

name: 'Hedy Lamarr',

imageUrl: 'https://i.imgur.com/yXOvdOSs.jpg',

imageSize: 90,

};

return (

<>

<h1>{user.name}</h1>

<img

className="avatar"

src={user.imageUrl}

alt={'Photo of ' + user.name}

style={{

width: user.imageSize,

height: user.imageSize

}}

/>

</>

);

}

#### Uso de expresiones condicionales

En React, no existe una sintaxis especial para escribir condiciones. En su lugar, utilizarás las mismas técnicas que usas al escribir código JavaScript regular. Por ejemplo, puedes usar una declaración if para incluir JSX de forma condicional:

let content;

if (isLoggedIn) {

content = <AdminPanel />;

} else {

content = <LoginForm />;

}

return (

<div>

{content}

</div>

);

Si prefieres un código más compacto, puedes utilizar el [operador ? condicional](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional_Operator). A diferencia de if, funciona dentro de JSX:

<div>

{isLoggedIn ? (

<AdminPanel />

) : (

<LoginForm />

)}

</div>

Cuando no necesites la rama else, puedes también usar la [sintaxis lógica &&](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_AND#short-circuit_evaluation), más breve:

<div>

{isLoggedIn && <AdminPanel />}

</div>

#### Renderizado de listas

Cuando tienes una lista y quieres usarla para tu código JSX , la forma más común de hacerlo es con la función map que devolverá JSX. Raramente necesitarás otro tipo de bucle que no sea este. A continuación, puedes ver cómo funciona.

const animals = ["Perro", "Gato", "Conejo", "Ratón", "Caballo"];

return (

<ul>

{animals.map((animal) => (

<li>{animal}</li>

))}

</ul>

);

Es importante introducir el concepto de key den los elementos renderizados de una lista para que evitar problemas de renderizao.

const animals = const animals = [

{ id: 1, animal: "Perro" },

{ id: 2, animal: "Gato" },

{ id: 3, animal: "Conejo" },

{ id: 4, animal: "Ratón" },

{ id: 5, animal: "Caballo" }

];

return (

<ul>

{animals.map(item => (

<li key={item.id}>{item.animal}</li>

))}

</ul>

);

Otra forma de iterar sobre listas podría ser esta : puedes primero usar una variable para almacenar los resultados del map y luego renderizar el contenido de la variable.

const mostrarAnimales = animals.map(item => (

<li key={item.id}>{item.animal}</li>

));

return <ul>{mostrarAnimals}</ul>;

### Gestión de eventos

En React, la interacción con el usuario se logra principalmente a través de la gestión de eventos. Cuando un usuario interactúa con un elemento en la interfaz de usuario (por ejemplo, hace clic en un botón), React dispara un evento que podemos capturar y manejar en nuestro código.

**Sintaxis básica**

Para agregar un manejador de eventos a un elemento en JSX, utilizamos el nombre del evento (en minúsculas y con on al principio) como atributo del elemento. El valor de este atributo es una función que se ejecutará cuando el evento ocurra.

import React from 'react';

function MiComponente() {

const handleClick = () => {

console.log('¡Hiciste clic!');

};

return (

<button onClick={handleClick}>Haz clic aquí</button>

);

}

En este ejemplo:

* **onClick** es el manejador de eventos para el evento de clic.
* **handleClick** es la función que se ejecutará cuando se haga clic en el botón.

**Pasando argumentos a los manejadores de eventos**

Podemos pasar argumentos adicionales a los manejadores de eventos, como el evento en sí, para obtener información sobre el evento.

const handleClick = (event) => {

console.log(event.target.value); // Obtiene el valor del elemento

};

<input type="text" onChange={handleClick} />

**Eventos comunes**

Algunos de los eventos más comunes en React son:

* onClick: Se dispara cuando se hace clic en un elemento.
* onChange: Se dispara cuando cambia el valor de un input (textos, selects, etc.).
* onSubmit: Se dispara cuando se envía un formulario.
* onMouseOver, onMouseOut: Se disparan cuando el ratón entra o sale de un elemento.

onKeyDown, onKeyUp: Se disparan cuando se presiona o se suelta una tecla.

**Prevención del comportamiento por defecto**

Algunos eventos tienen un comportamiento por defecto (por ejemplo, el evento submit de un formulario recarga la página). Para prevenir este comportamiento, podemos utilizar el método preventDefault del evento.

const handleSubmit = (event) => {

event.preventDefault();

// Aquí puedes realizar tus acciones

};

<form onSubmit={handleSubmit}>

{/\* ... \*/}

</form>

#### Tipos de Variables en Componentes React

En React, los componentes pueden utilizar diferentes tipos de variables para almacenar y gestionar datos. La elección del tipo de variable dependerá del contexto y de cómo se necesite que los datos se comporten.

### Hooks react

**Props (Propiedades)**

* **Definición:** Son datos que se pasan a un componente desde su componente padre.
* **Características:**
  + Son inmutables dentro del componente hijo.
  + Sirven para personalizar la apariencia y el comportamiento de un componente.
  + Se utilizan para pasar datos estáticos o que no cambian con frecuencia.
* **Ejemplo:**

function Saludo(props) {

return <h1>Hola, {props.nombre}!</h1>;

}

<Saludo nombre="Juan" />

**State (Estado)**

* **Definición:** Representa datos internos de un componente que pueden cambiar con el tiempo en respuesta a acciones del usuario o eventos.
* **Características:**
  + Se gestiona mediante el hook useState o el método setState en clases.
  + Es mutable dentro del componente.
  + Se utiliza para almacenar datos que cambian con frecuencia, como el valor de un input o la visibilidad de un modal.
* **Ejemplo:**

import { useState } from 'react';

function Contador() {

const [contador, setContador] = useState(0);

return (

<div>

<p>Contador: {contador}</p>

<button onClick={() => setContador(contador + 1)}>Incrementar</button>

</div>

);

}

**Variables locales**

* **Definición:** Variables declaradas dentro de una función o un bloque de código.
* **Características:**
  + Son específicas de un ámbito determinado y no se comparten entre componentes.
  + Se utilizan para almacenar valores temporales o resultados de cálculos.
* **Ejemplo:**

function MiComponente() {

const mensaje = 'Hola, mundo';

return <p>{mensaje}</p>;

}

**Contexto (Context API)**

* **Definición:** Mecanismo para compartir datos entre componentes que están muy separados en la jerarquía de componentes sin pasar props explícitamente en cada nivel.
* **Características:**
  + Crea un contexto global al que pueden suscribirse los componentes.
  + Es útil para compartir datos como temas, idioma o información de autenticación.
* **Ejemplo**

import { createContext, useContext } from 'react';

const ThemeContext = createContext(null);

export default function MyApp() {

return (

<ThemeContext.Provider value="dark">

<Form />

</ThemeContext.Provider>

)

}

function Form() {

return (

<Panel title="Welcome">

<Button>Sign up</Button>

<Button>Log in</Button>

</Panel>

);

}

function Panel({ title, children }) {

const theme = useContext(ThemeContext);

const className = 'panel-' + theme;

return (

<section className={className}>

<h1>{title}</h1>

{children}

</section>

)

}

function Button({ children }) {

const theme = useContext(ThemeContext);

const className = 'button-' + theme;

return (

<button className={className}>

{children}

</button>

);

}

**useRef**

* **¿Qué es?** useRef es un hook de React que se utiliza para crear y persistir referencias a DOM o valores.
* **Cuándo usarlo:**
  + Para acceder a elementos DOM directamente.
  + Para almacenar valores que no necesitan re-renderizar el componente.
* **Ejemplo:**

import React, { useRef } from 'react';

function FormularioAutoenfocado() {

const nombreInput = useRef(null);

useEffect(() => {

nombreInput.current.focus();

}, []);

return (

<form>

<input type="text" ref={nombreInput} placeholder="Ingrese su nombre" />

<button type="submit">Enviar</button>

</form>

);

}

**useMemo**

* **¿Qué es?** useMemo es un hook de React que se utiliza para memorizar el resultado de una función.
* **Cuándo usarlo:**
  + Para optimizar el rendimiento de componentes que realizan cálculos costosos.
  + Para evitar re-renderizados innecesarios.

import React, { useMemo } from 'react';

function Calculadora(props) {

const { numero1, numero2 } = props;

const resultado = useMemo(() => {

return numero1 + numero2;

}, [numero1, numero2]);

return (

<div>

<p>Resultado: {resultado}</p>

</div>

);

}

En este ejemplo, useMemo se utiliza para calcular la suma de numero1 y numero2 de forma eficiente. El resultado se memoriza, por lo que solo se recalcula si los valores de numero1 o numero2 cambian.

**UseEffects**

El hook useEffect es una utilidad en React paraincluir comportamientos cuando sucede algo en la aplicación:

import React, { useState, useEffect } from 'react';

function UserList() {

const [users, setUsers] = useState([]);

useEffect(() => {

const fetchUsers = async () => {

const response = await fetch('https://api.example.com/users');

const data = await response.json();

setUsers(data);

};

fetchUsers();

}, []); // El array vacío indica que el efecto se ejecuta solo una vez, al montar el componente

return (

<div>

<h2>Lista de Usuarios</h2>

<ul>

{users.map(user => (

<li key={user.id}>{user.name}</li>

))}

</ul>

</div>

);

}

**Explicación:**

1. **useState:** Se utiliza para crear el estado users, que almacenará la lista de usuarios.
2. **useEffect:**
   * Se ejecuta después de cada renderizado, pero solo se ejecuta el efecto una vez al montar el componente, gracias al array vacío como segundo argumento.
   * Dentro del efecto, se define una función asíncrona fetchUsers que realiza la petición a la API y actualiza el estado users con los datos obtenidos.
3. **Renderizado:**
   * El componente renderiza la lista de usuarios utilizando el estado users

Otro ejemplo:

import React, { useEffect } from 'react';

function UserProfile({ name }) {

useEffect(() => {

document.title = `Perfil de ${name}`;

}, [name]); // El efecto se ejecuta cuando cambia el valor de 'name'

return (

<div>

<h1>Perfil de {name}</h1>

</div>

);

}

* **Array de dependencias:** En este caso, el array de dependencias contiene [name]. Esto significa que el efecto se ejecutará cada vez que el valor de la prop name cambie.
* **Actualización del título:** Dentro del efecto, se actualiza el título del documento utilizando document.title para reflejar el nuevo nombre del usuario.

### Uniendo todo lo anterior

Vamos a crear un pequeño tutorial de React para compartir datos entre componentes, utilizando el ejemplo de un carrito de compras.

En una aplicación React, a menudo necesitamos compartir datos entre diferentes componentes. Por ejemplo, en un carrito de compras, queremos que la lista de productos agregados sea visible en todas las páginas de la aplicación.

**Métodos para compartir datos en React:**

* **Props:** Pasar datos de un componente padre a un componente hijo.
* **State:** Mantener el estado local dentro de un componente.
* **Context API:** Crear un contexto para compartir datos a través de la jerarquía de componentes.

#### Ejemplo: Carrito de compras con Context API

Para este ejemplo, utilizaremos Context API, ya que es una solución sencilla y efectiva para compartir datos entre componentes relacionados.

1.Crear un contexto:

import React, { createContext, useState } from 'react';

const CartContext = createContext();

export const CartProvider = ({ children }) => {

const [cartItems, setCartItems] = useState([]);

const addToCart = (item) => {

setCartItems([...cartItems, item]);

};

const removeFromCart = (itemId) => {

setCartItems(cartItems.filter(item => item.id !== itemId));

};

return (

<CartContext.Provider value={{ cartItems, addToCart, removeFromCart }}>

{children}

</CartContext.Provider>

);

};

1. Usar el contexto en los componentes:

import React, { useContext } from 'react';

import { CartContext } from './CartContext';

function ProductList() {

const { cartItems, addToCart } = useContext(CartContext);

const products = [

{ id: 1, name: 'Producto 1', price: 10 },

{ id: 2, name: 'Producto 2', price: 20 },

// ... más productos

];

return (

<div>

<h2>Lista de productos</h2>

<ul>

{products.map(product => (

<li key={product.id}>

{product.name} - ${product.price}

<button onClick={() => addToCart(product)}>Agregar al carrito</button>

</li>

))}

</ul>

</div>

);

}

function Cart() {

const { cartItems } = useContext(CartContext);

return (

<div>

<h2>Carrito de compras</h2>

<ul>

{cartItems.map(item => (

<li key={item.id}>{item.name}</li>

))}

</ul>

</div>

);

}

3. Envolver la aplicación con el proveedor de contexto:

import React from 'react';

import { CartProvider } from './CartContext';

import ProductList from './ProductList';

import Cart from './Cart';

function App() {

return (

<CartProvider>

<ProductList />

<Cart />

</CartProvider>

);

}

**Explicación:**

1. **Creamos un contexto:** Definimos un contexto CartContext y un proveedor CartProvider que envuelve toda la aplicación.
2. **Almacenamos el estado del carrito:** En el proveedor, utilizamos useState para almacenar la lista de productos en el carrito.
3. **Proporcionamos funciones para modificar el carrito:** Las funciones addToCart y removeFromCart permiten agregar y eliminar productos del carrito.
4. **Utilizamos el contexto en los componentes:** Los componentes que necesitan acceder al carrito pueden utilizar el hook useContext para obtener los valores del contexto.

#### Ejemplo: Noticias -props

Estructura componente padre

import React, { useState, useEffect } from 'react';

import NewsItem from './NewsItem'; // Componente hijo para cada noticia

const NewsFeed = () => {

const [news, setNews] = useState([]);

const [savedNews, setSavedNews] = useState([]);

// Función para obtener las noticias (simulando un servicio)

const fetchNews = async () => {

const response = await fetch('https://api.example.com/news');

const data = await response.json();

setNews(data);

};

useEffect(() => {

fetchNews();

}, []);

// Función para guardar una noticia

const saveNewsItem = (newsItem) => {

setSavedNews([...savedNews, newsItem]);

};

return (

<div>

<h1>Noticias</h1>

<div>

{news.map((item) => (

<NewsItem key={item.id} newsItem={item} onSave={saveNewsItem} />

))}

</div>

<h2>Noticias Guardadas</h2>

<div>

{savedNews.map((item) => (

<NewsItem key={item.id} newsItem={item} />

))}

</div>

</div>

);

};

export default NewsFeed;

Estructura componente hijo

import React from 'react';

const NewsItem = ({ newsItem, onSave }) => {

return (

<div>

<h3>{newsItem.title}</h3>

<p>{newsItem.description}</p>

<button onClick={() => onSave(newsItem)}>Guardar</button>

</div>

);

};

export default NewsItem;

* **Componente Padre (NewsFeed):**
  + Mantiene el estado de las noticias obtenidas y las noticias guardadas por el usuario.
  + Utiliza useEffect para obtener las noticias al cargar el componente.
  + Pasa la función saveNewsItem al componente hijo para permitir guardar noticias.
  + Renderiza una lista de noticias y una lista de noticias guardadas.
* **Componente Hijo (NewsItem):**
  + Muestra los detalles de una noticia individual.
  + Tiene un botón para guardar la noticia, que llama a la función onSave pasada desde el componente padre.

**3. Funcionamiento:**

1. Al cargar la aplicación, se obtiene un listado de noticias desde el servicio.
2. Cada noticia se renderiza en un componente NewsItem.
3. Al hacer clic en el botón "Guardar" de una noticia, se llama a la función saveNewsItem del componente padre, agregando la noticia a la lista de noticias guardadas.
4. La lista de noticias guardadas se renderiza por separado.

**Lectura ficheros json**

JSON, que significa JavaScript Object Notation (Notación de Objetos de JavaScript), es un formato de texto ligero para el intercambio de datos. Es una forma sencilla y legible por humanos de organizar información, similar a un diccionario o lista. Se basa en la sintaxis de los objetos en JavaScript, pero puede ser utilizado por muchos otros lenguajes de programación.

¿Para qué se utilizan los archivos JSON?

Los archivos JSON se utilizan ampliamente en el desarrollo web y de aplicaciones para:

* **Almacenar datos:** Se pueden guardar datos de configuración, datos de usuarios, resultados de consultas a bases de datos, etc.
* **Intercambiar datos entre sistemas:** Son muy usados para enviar/recibir datos entre un servidor y un cliente (por ejemplo, en una aplicación web).
* **Configurar aplicaciones:** Muchos programas utilizan archivos JSON para almacenar sus configuraciones.

Un archivo JSON consta de pares clave-valor, separados por comas y encerrados entre llaves. Las claves son cadenas de texto y los valores pueden ser:

* Cadenas: Texto encerrado entre comillas dobles.
* Números: Enteros o decimales.
* Booleanos: true o false.
* Null: Representa un valor nulo.
* Objetos: Otros objetos JSON encerrados entre llaves.
* Arrays: Listas de valores encerrados entre corchetes.

{

"nombre": "Juan",

"edad": 30,

"ciudad": "Madrid",

"esEstudiante": false

}

Ejemplo de array

[

"manzana",

"banana",

"pera"

]

Ejemplo de artículos

[

{

"id": 1,

"nombre": "Árbol de Navidad artificial",

"descripcion": "Árbol de Navidad artificial de 1.80 metros con luces LED",

"precio": 99.99

},

{

"id": 2,

"nombre": "Guirnalda luminosa",

"descripcion": "Guirnalda luminosa con 100 luces LED de colores",

"precio": 19.99

}

]

La lectura de ficheros json en java se realiza con la función fetch

fetch("https://reqres.in/api/users")

.then(response => response.json())

.then(data => {

console.log(data);

})

.catch(error => {

console.error('Error al obtener los datos:', error);

});

Y en un componente en React

import React, { useState, useEffect } from 'react';

function App() {

const [users, setUsers] = useState([]);

useEffect(() => {

const fetchUsers = async () => {

const response = await fetch('https://reqres.in/api/users');

const data = await response.json();

setUsers(data.data);

};

fetchUsers();

}, []);

return (

<div>

<h1>Lista de Usuarios</h1>

<ul>

{users.map(user => (

<li key={user.id}>

{user.first\_name} {user.last\_name}

</li>

))}

</ul>

</div>

);

}

export default App;