

Índice

- Índice
- ¿Qué es WordPress?
- Despliegue
- Introducción
- Comenzando con React
- Componentes
- ▼ Mi primera aplicación con React.
 - Creación de un proyecto
 - Estructura de directorios
 - ▼ Creando el primer componente
 - Comunicación con componentes

¿Qué es WordPress?

El término CMS proviene del inglés Content Management System, que significa Sistema de Gestión de Contenidos, un sistema online que nos permite poner en marcha un sitio web de forma práctica y rápida. Su gran ventaja es la posibilidad de administrar contenidos dinámicos de forma sencilla, es decir, mantener un blog, un ecommerce o cualquier otro tipo de página web que demande una actualización constante.

Imagínate por ejemplo que trabajas en un periódico o una revista. Tu audiencia siempre está buscando noticias de último momento y temas de actualidad, por lo que tú y tu equipo tenéis producir contenidos nuevos constantemente. Este proceso de creación puede ser complejo: para cada nuevo contenido, es necesario crear la estructura técnica completa de la página y cada actualización manual lleva su tiempo.

En tal escenario, ¿te imaginas lo caro y tardado que sería este trabajo? Sería insostenible como negocio, en términos de tiempo y dinero.

Además, suponiendo que pudieras mantener el proyecto en marcha, la necesidad de contar con profesionales altamente capacitados y con los conocimientos técnicos necesarios también podría limitar en gran medida la aparición de nuevos negocios.

Para solucionar este tipo de problema estructural y operativo, se creó el CMS.

CMS son las siglas de "Content Management System" ("sistema de gestión de contenidos"). Este tipo de herramientas te ayudan a gestionar tu sitio web, hacer cambios y subir contenidos, todo ello de manera fácil e intuitiva y sin necesidad de tener conocimientos de código.

El CMS tiene tres funciones principales: Crear sitios web, gestionarlos y mantenerlos y darnos herramientas para poder administrarlos. Hoy en día existen muchos tipos diferentes de CMS en el mercado pero los más conocidos son:

- *WordPress*
- Drupal
- Joomla
- Wix
- Magento

Y entre tantas opciones, ¿Por qué WordPress? Principalmente por sus características:

- Facilidad de uso: presenta una interfaz amigable que no exige tener conocimientos técnicos elevados de la web.

- Personalización: existen numerosos diseños (más de 10.000 plantillas) y plugings que nos permiten configurar el sitio web tanto a nivel estético como de funcionalidad.
- Amplia comunidad que da soporte a nivel de recursos (información, soporte, dudas...). Además es código abierto.
- SEO: está diseñado teniendo en cuenta las recomendaciones para mejorar el posicionamiento de la web
- Actualizaciones y seguridad: el equipo técnico está en constante mejora de su seguridad.

Despliegue

- Comprobar si Node se ha instalado correctamente

```
> npm --version  
9.5.1
```

- Crear proyecto Next.js

```
> npx create-next-app
```

-


Introducción

React es una librería para crear interfaces de usuario en el mundo web. [EN OBRRAS: Ventajas de React]

```
/*Ejemplo de código escrito en React: insertamos un título de nivel 1 con contenido H  
ReactDOM.render( <h1>HOLA MUNDO</h1>, document.getElementById('principal'));
```

Comenzando con React

1. Creamos en VSCode un nuevo archivo .html

 Gracias a los Emmet snippets recuerda que teniendo un archivo con extensión .html sólo tienes que teclear ! o html5 y te generará la estructura básica de un documento html.

2. Añadimos a nuestro html la librería React y babel, que nos va a ayudar a hacer la traducción de etiquetado html a elementos directamente. Además insertaremos también una capa para posteriormente crear con React un elemento.

```

<body>
  <!--Nuestro marcado html-->
  <div id="principal"></div>

  <!--Librería React-->
  <script src="https://unpkg.com/react@17/umd/react.production.min.js" crossorigin="anonymous"></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js" crossorigin="anonymous"></script>

  <!--Librería babel-->
  <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
</body>

```

3. Insertamos al etiqueta script donde vamos a escribir nuestras primeras instrucciones en React.

```

...
<!--Nuestro marcado html-->
<div id="principal"></div>

<script type="text/babel">
  /*Recuperamos el elemento donde quiero insertar el nuevo contenido*/
  const principal = document.querySelector("#principal");

  /*Creamos un elemento título de nivel 1 que contenga el texto ¡Hola mundo!*/
  const saludo = <h1>Hola mundo</h1>;

  /*Le decimos a React que renderice el elemento h1 dentro del contenedor principal*/
  ReactDOM.render(saludo,principal);
</script>

<!--Librería React-->
...

```

4. Observamos el resultado en el navegador

 Hola mundo

Componentes

Antes de comenzar a hacer la primera página hay que aclarar el concepto de componentes. Los componentes son uno de los conceptos principales dentro de React. Podemos verlos como bloques de código, piezas de la interfaz. Van a ser reutilizables. Juntando todas estas piezas diseñaremos la interfaz.

Conceptualmente van a ser como las funciones JS, van a aceptar entradas y retornar elementos de React.

Componentes

Los componentes tienen un estado. El estado va a ser cómo se encuentra la información en un determinado momento dentro de nuestro componente, por ejemplo, el estado nos va a ayudar a recuperar los valores de dentro de un input, que estarán almacenados dentro del estado del componente formulario. Además tienen otro tipo de características muy interesantes:

Componentes

Mi primera aplicación con React.

Creación de un proyecto

1. Para crear una aplicación con React debemos ejecutar el siguiente comando, que nos creará un proyecto react de nombre ejemplo-app (si quieres puedes consultar la [documentación oficial](#))

```
npx create-react-app título-app
```

El proceso puede tardar unos minutos. Una vez finalizado veremos en consola un mensaje similar al siguiente:

 Creación proyecto React

2. Entramos al directorio título-app y lanzamos el servidor.

```
cd título-app  
npm start
```

3. Se abrirá el navegador Chrome con una aplicación por defecto

 Proyecto por defecto React

Estructura de directorios

Si abrimos con VSCode el directorio ejemplo-app veremos que se han creado una serie de archivos y directorios

 Estructura de directorios

Esta estructura puede cambiar según las versiones pero los directorios y archivos fundamentales son:

- node_modules: contiene las librerías y paquetes necesarios para poder desarrollar con React. Esas carpetas no debemos manipularlas directamente.
- public: aparecen los recursos de la página web que estamos visualizando. Tan solo el archivo index.html es el que nos interesa. El resto de archivos se utilizan para aplicaciones PWA

(aplicaciones web progresivas)

- index.html: En el index.html creado por defecto hay una serie de etiquetas y atributos que merecen explicación
 - El atributo lang de la etiqueta html aparece por defecto en inglés, podríamos cambiarlo a español.
 - En la etiqueta link que adjunta el favicon hace uso de la variable %PUBLIC_URL% que nos va a asegurar la independencia de la url.
 - La etiqueta meta name=theme-color... No tiene mucho sentido puesto que será utilizado pro aplicaciones PWA. Podríamos eliminarla. Al igual que el link a las dos imágenes logo y los comentarios.
 - La etiqueta noscript... Es una etiqueta que se usa para indicar que el navegador no tiene activado JavaScript. Actualmente es raro encontrarse con esta situación pero no está de más ponerlo.
 - Puedes visualizar cómo quedaría el fichero index.html con los cambios propuestos [Aquí](#) o dejarlo tal y como estaba.
- .gitignore y [README.md](#): Como ya sabes es un archivo relacionado con el control de versiones.
- package.json: Este archivo nos servirá para ejecutar los scripts... En principio no vamos a editarlo.
- src: contiene una serie de archivos relacionados con la página por defecto

Creando el primer componente

1. Abrimos nuestro proyecto y vaciamos la carpeta src
2. Dentro de ese directorio creamos un primer archivo llamado index.js y otro PrimerComponente.js. Es importante usar nomenclatura UpperCamelCase pues es parte del estándar que los componentes se nombren así.

 **Nota:** Recuerda tener lanzado el servidor (npm start) e ir observando los cambios.

1. Debemos importar React, ReactDOM, las posibles hojas de estilo que más adelante haremos y los componentes que vamos a crear. Para eso añadimos estas primeras líneas a nuestro archivo index.js


```

/*Importación de react*/
import React from 'react';
import ReactDOM from 'react-dom';

/*Importación de componentes*/
import Cabecera from './Cabecera';

/*Importación de ficheros css*/
import './index.css';

```

1. Vamos a crear nuestro primer componente funcional. Escribimos una función lambda y la exportamos en Cabecera.js. En esa función retornaremos los elementos que queremos que formen parte de este componente. Hay que ponerlo entre () y crear un fragment para agrupar todos los elementos en uno, ya que React sólo espera procesar un elemento.

```

import { Fragment } from "react";

const Cabecera = () =>
{
  return (
    <Fragment >
      <h1>Fito y Fitipaldis</h1>
      <ul>
        <li>Cada vez cadáver</li>
        <li>Lo más lejos a tu lado</li>
        <li>Antes de que cuente diez</li>
        <li>Por la boca vive el pez</li>
        <li>Huyendo conmigo de mí</li>
      </ul>
    </Fragment>
  );
}

export default Cabecera;

```

5. En index.js renderizamos con React el componente

```

/*El primer parámetro que recibe el método render en este caso es el componente. El c
ReactDOM.render(<Cabecera />,document.querySelector("header"));


```

6. Creamos un nuevo archivo index.css en el directorio src. Añadimos una serie de estilos.

```
body
{
  min-height: 100vh;
  max-width: 100vw;
  background-color: #21232a;
}
header
{
  background-color: cornflowerblue;
  color: white;
}
ul
{
  display: flex;
  justify-content: space-around;
  list-style: none;
}
```

👁️ Puedes observar cómo se han quedado los ficheros dentro del directorio título-app

Ahora mismo nuestra página web tendrá un aspecto similar al siguiente:

 Imagen cabecera

Comunicación con componentes

En el ejemplo anterior, mi cabecera siempre contiene la información de Fito y Fitipaldis, pero ¿Y si quiere que mi componente sea dinámico y según la página que esté creando me muestre una información u otra? Vamos a ver cómo podemos hacer esto.

Coge el ejemplo anterior y creamos un nuevo proyecto que vamos a llamar títulos2-app.

Simplemente copia el directorio y desde consola ejecuta `npm start` dentro del nuevo directorio.

1. Vamos a modificar el fichero `index.js`. Usaremos las llamadas props, que no son más que las propiedades de un componente, que nos van a dar capacidad de reutilización y poder mejorar la funcionalidad. Para añadir una propiedad, en el fichero `index.js`, en el componente cabecera pondremos un atributo `grupo="Izal"`.

```

/*Importación de react*/
import React from 'react';
import ReactDOM from 'react-dom';

/*Importación de componentes*/
import Cabecera from './Cabecera';

/*Importación de ficheros css*/
import './index.css';

/*El primer parámetro que recibe el método render en este caso es el componente. El c
ReactDOM.render(<Cabecera grupo="Izal" discos="Hogar;Autoterapia;Copacabana;Agujeros

```

2. Abrimos el archivo Cabecera.js y recuperamos el dato pasado usando los props, que es un objeto donde vamos a poder almacenar todas las propiedades del componente. Para acceder a una propiedad usaremos el operador . y el nombre de la propiedad.


```

const Cabecera = (props)=>
{
  let discos = props.discos.split(";");

  return (
    <Fragment >
      <h1>{props.grupo}</h1>
      <ul>
        <li>{discos[0]}</li>
        <li>{discos[1]}</li>
        <li>{discos[2]}</li>
        <li>{discos[3]}</li>
        <li>{discos[4]}</li>
      </ul>
    </Fragment>
  );
}

```

3. Ahora mismo podemos ver que tenemos el mismo componente, cabecera, pero que hemos cambiado el contenido simplemente con un parámetro.

 Imagen cabecera