



**Word-
Camp**
Zaragoza
2023

Programando un bloque Gutenberg en React desde cero.

Joaquín Ruiz
@JokiRuizLite - jokiruiz.com

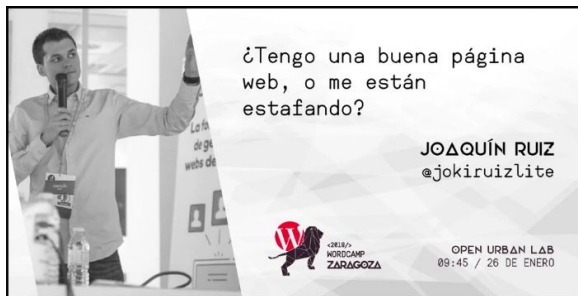
#WCZGZ23

Hola! Soy Joaquín Ruiz (@jokiruilite)

Ingeniero Informático con más de 12 años de experiencia en desarrollo web, ecommerce y optimización de sistemas.

CTO en FLAT IOI Digital Business

Profesor de Grado Superior en SANVALERO  GRUPO SANVALERO



¿Qué vamos a ver hoy?

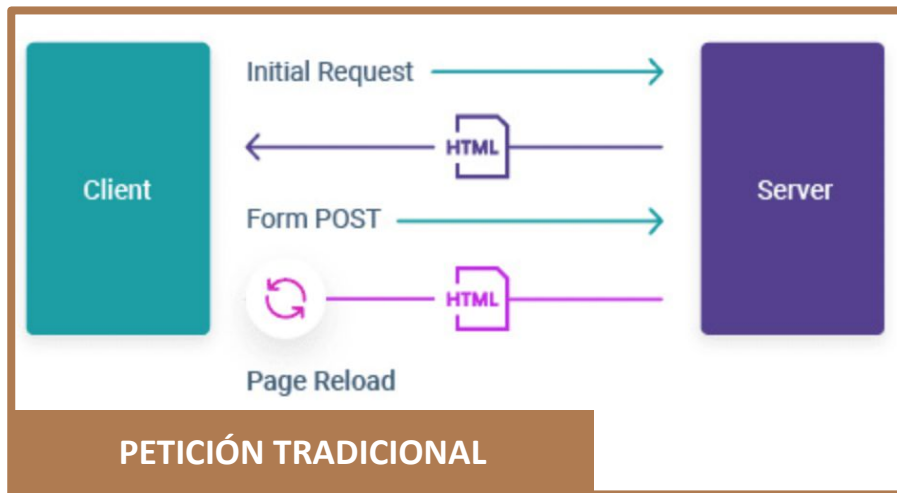
Los básicos de una SPA como React y cómo registrar un bloque Gutenberg

1. ¿Qué es una SPA?
2. Introducción a React... ¿Cómo funciona una SPA?
3. Registrando un bloque Gutenberg (taller práctico)

1. ¿Qué es una SPA?

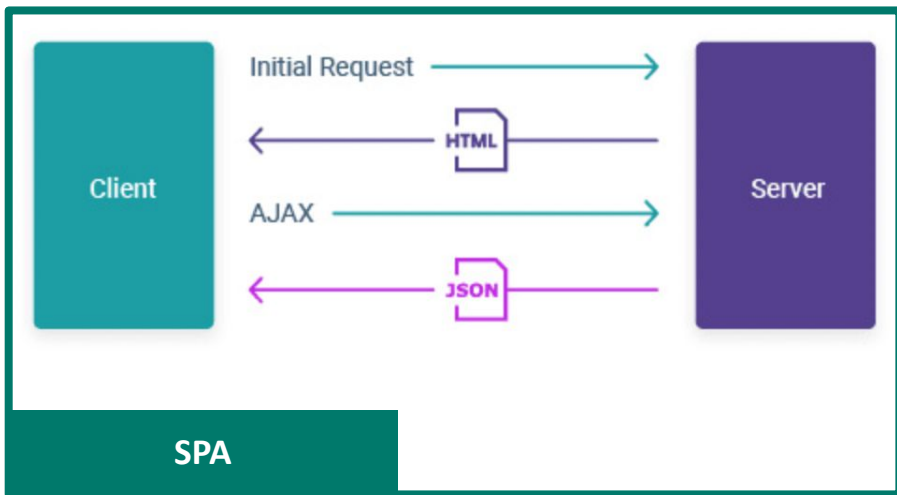
¿Cómo funciona un sitio web tradicional?

Cuando escribimos la URL de un sitio web o interactuamos con cualquier botón o enlace, estamos mandando **peticiones (o HTTP Request)** a un **servidor**, que nos contesta generalmente con **documentos HTML completos** que se renderizan en nuestro navegador.

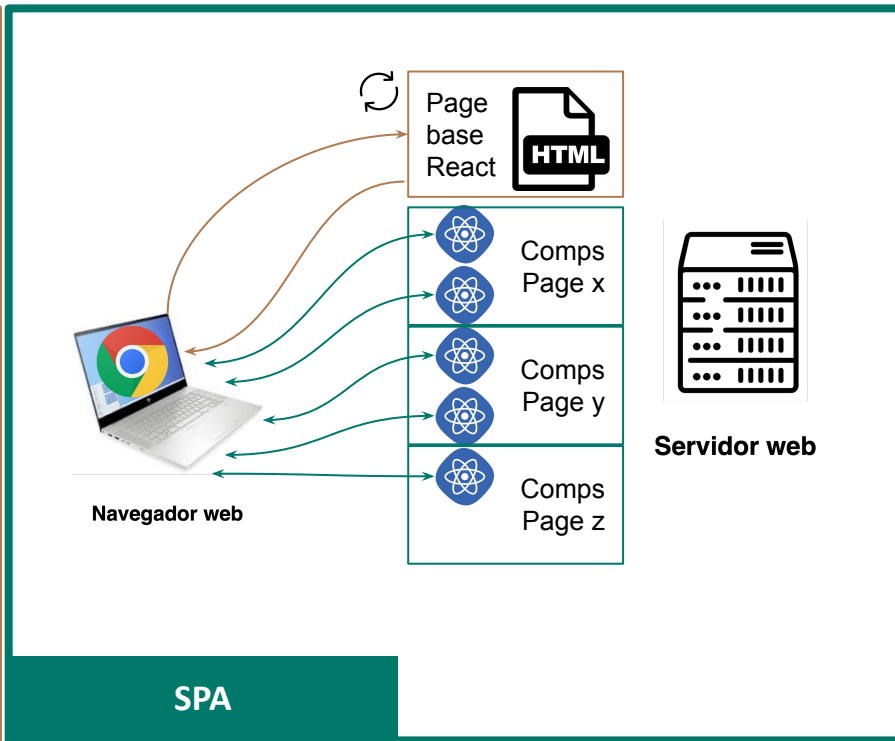
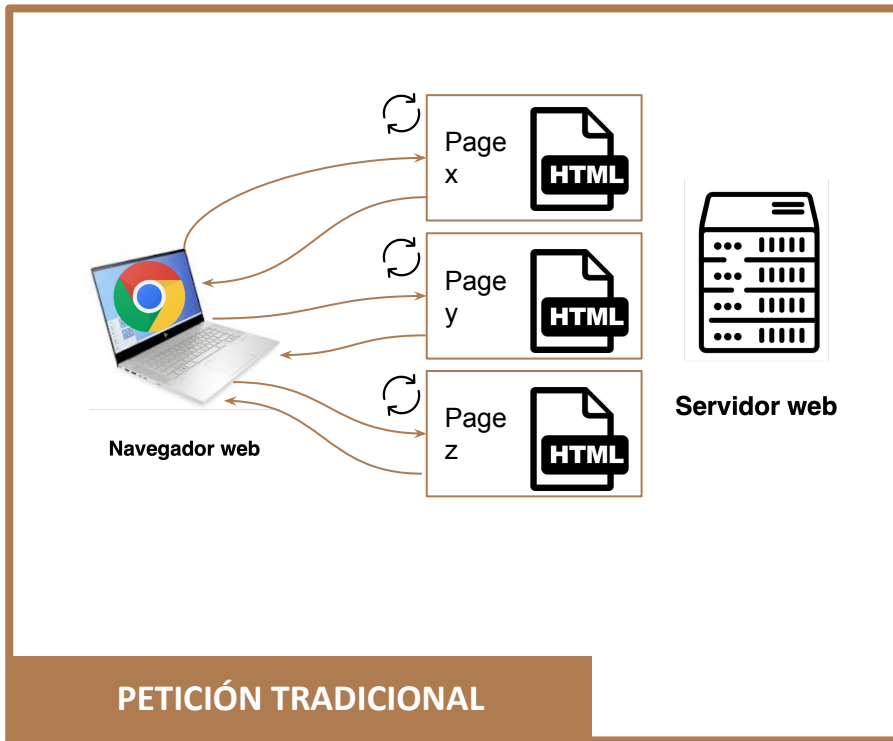


SPA: Cambio de paradigma

Una Single Page Application (comúnmente abreviada como SPA), es un sitio web con solo un documento **HTML que se carga una única vez**, cargando el contenido y simulando la navegación entre páginas **a través de JavaScript** (mediante peticiones **AJAX**), en vez de hacer más peticiones completas de HTML al servidor.



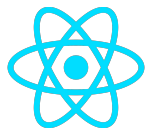
SPA: Cambio de paradigma



SPA: Beneficios

- **Obtenemos toda la información de forma asíncrona**
 - Mejor rendimiento y carga
- **Mejor Usabilidad (UX)**
 - El usuario no tiene que esperar a que se recargue la página
- **Un mayor control sobre el flujo de navegación del usuario**
 - Mayor capacidad y facilidad para crear elementos reactivos
- **Plataformas más ligeras (menos computación a nivel de servidor)**
 - El servidor sólo nos envía información, la capa de presentación la realiza el navegador

2. Introducción a React



React es la librería para desarrollar interfaces y aplicaciones de tipo SPA más utilizada en la actualidad.

A diferencia del **JavaScript tradicional** o **jQuery**, que cuando seleccionamos un componente de nuestro sitio web usamos el **Browser DOM**, React utiliza **Virtual DOM**, que está almacenado en memoria, y así podemos recalcular de forma inteligente y reactiva todos los cambios que se producen.

Browser DOM

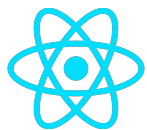
```
<div class="commentBox">  
</div>
```

```
let commentBox= document.getElementsByClassName("commentBox");  
commentBox.html("WordPress Zaragoza!");
```

```
registerBlockType('jokiruiz/misco-block', {  
  
  edit : (props) => {  
    const { attributes: { name }, setAttributes } = props ;  
  
    return (  
      <div className="commentBox">  
        {name}  
      </div>  
    )  
  }  
});
```

Virtual DOM

React



El Componente

Un **componente** es la unidad básica para crear elementos en el Virtual DOM, que incluye una **capa visual** (HTML+CSS) y una **capa lógica** (el comportamiento).

Los componentes se declaran como una función devuelve un **código JSX**.

```
return (  
  <div className="commentBox">  
    {name}  
  </div>  
)
```

Props

Los componentes en React, pueden recibir diferentes **valores de entrada**.

Este componente podría recibir un texto diferente en función de dónde queramos utilizarlo. **Estos valores de entrada son lo que llamamos props.**

Estos props pueden ser **cambiados en cualquier momento, y actualizados** en tiempo real sin tener que programar dicha actualización.

```
return (  
  <div className="commentBox">  
    {name}  
  </div>  
)
```

```
const { attributes: { name }, setAttributes } = props ;  
return (  
  <div className="commentBox">  
    {name}  
  </div>  
)
```

States

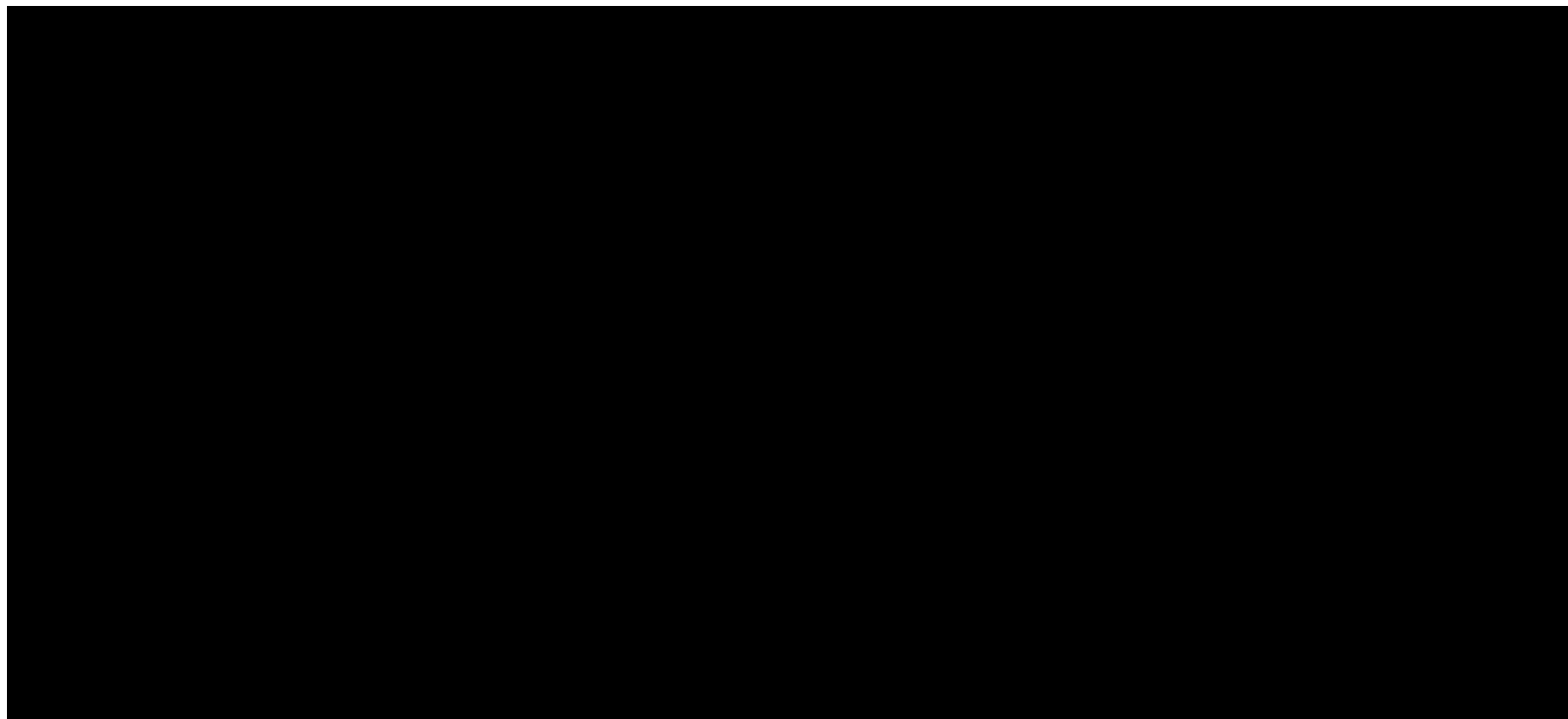
El state es el **conjunto de datos** que maneja el componente a nivel interno.

Esta información no es exterior, sino que es controlada directamente por el propio componente en **función de la lógica programada**.

```
const { attributes: { name }, setAttributes } = props ;
const changeName = (value) => {
  setAttributes({name:value})
};
return (
  <div className="misc-block">
    <div className="editor">
      <TextControl
        label="Introduce un comentario"
        value={ name }
        onChange={ changeName }
      />
    </div>
    <div className="commentBox">
      {name}
    </div>
  </div>
)
```

Voilà, un bloque Gutenberg!!

No era tan difícil no? ;)





3. Registrando un bloque Gutenberg (taller práctico)

Creando un Plugin en WordPress

/wordcamp-gutenberg-block/



wordcamp-gutenberg-block.php

Declarar el Plugin “wordcamp-gutenberg-block”
(nombre, descripción, etc)

Incluir la clase que añadirá la lógica del plugin

Registrando los Scripts y el bloque

/wordcamp-gutenberg-block/

 wordcamp-gutenberg-block.php

 /inc/create-block.php

En este fichero create-block:

- Registramos el bloque Gutenberg mediante ***register_block_type()***
https://developer.wordpress.org/reference/functions/register_block_type/
- Añadimos los ficheros de React (JS) mediante ***wp_enqueue_script()***
https://developer.wordpress.org/reference/functions/wp_enqueue_script/



Añadiendo dependencias

/wordcamp-gutenberg-block/



wordcamp-gutenberg-block.php



/inc/create-block.php



package.json

El fichero *package.json*

- Se especifican las dependencias y librerías de Node para crear un bloque Gutenberg, es similar al fichero composer de PHP.

<https://developer.wordpress.org/block-editor/reference-guides/packages/packages-scripts/>
`npm install @wordpress/scripts --save-dev`

Escribiendo el bloque en React

/wordcamp-gutenberg-block/

 wordcamp-gutenberg-block.php

 /inc/create-block.php

 package.json

 /src/index.js

El fichero *index.js*:

- Importamos la función ***registerBlockType()*** de la librería `@wordpress/blocks` y con ella añadimos la lógica necesaria de nuestro bloque Gutenberg

```
import { registerBlockType } from '@wordpress/blocks'
```

<https://developer.wordpress.org/block-editor/reference-guides/block-api/block-registration/>

registerBlockType

```
registerBlockType("ID_BLOQUE", {
```

```
  title: "TITULO_BLOQUE",
```

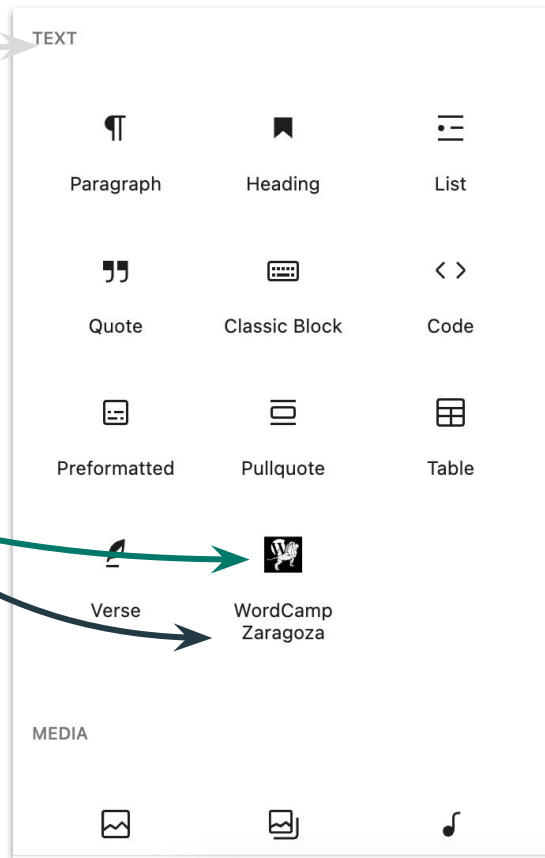
```
  description: "DESCRIPCION_BLOQUE",
```

```
  category: "text | media | ...",
```

```
  icon: "dashicon | <svg>...</svg>",
```

```
  (...)
```

```
})
```



registerBlockType

```
registerBlockType("ID_BLOQUE", {
```

```
(...)
```

```
attributes: {
```

```
  props necesarios
```

```
},
```

```
edit : (props) => {
```

```
  componente vista editor-backend
```

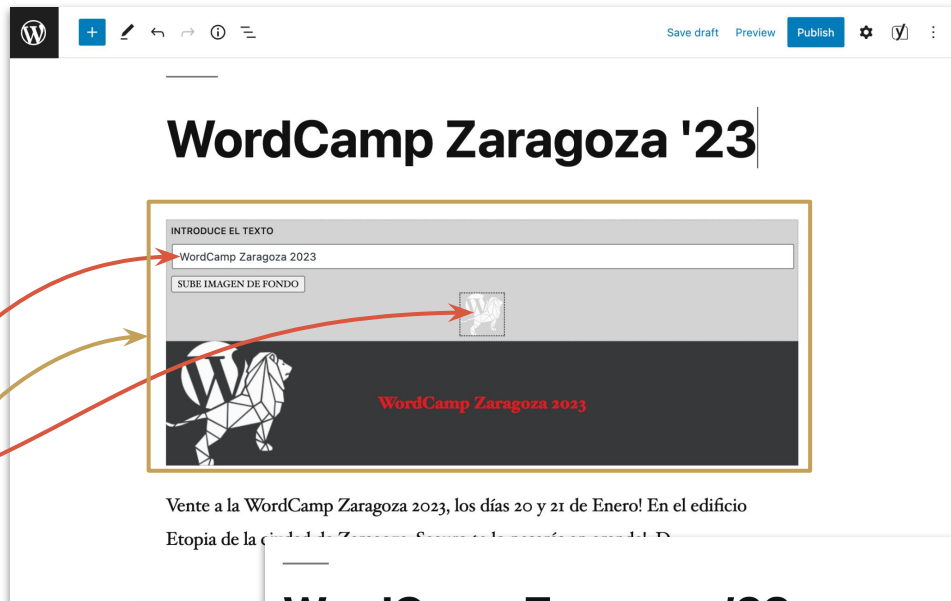
```
},
```

```
save : (props) => {
```

```
  componente vista frontend
```

```
}
```

```
})
```



Añadiendo estilos

/wordcamp-gutenberg-block/



wordcamp-gutenberg-block.php



/inc/create-block.php



package.json



/src/index.js



/src/editor.css



/src/index.css

- El fichero *editor.css* tendrá las clases que queramos incluir en el componente de backend
- El fichero *index.css* tendrá las clases que queramos incluir en el componente de frontend

Ambos habrá que añadirlos en “*inc/create-block.php*” mediante ***wp_enqueue_style()***

Compilando el bloque

Necesitaremos compilar nuestro fichero React mediante:

- `npm run start`
- `npm run build`

Estos comandos crean los ficheros finales minificados en la carpeta `/build/`, que es la que incluimos en WordPress

Dudas, preguntas!

