

HOLA REACT

Nuestro primer Hola React

1. Importamos las dependencias

```
import React from 'react';
import { createRoot } from 'react-dom/client';
```

2. Creamos un elemento de React

```
const element = React.createElement(
  'h1' ,
  { id: 'hola' },
  'Hola React'
);
```

3. Rendereamos la aplicación

```
const container = document.querySelector('#root');
const root = createRoot(container);
root.render(element);
```

DEPENDENCIAS

1. Importamos las dependencias

```
import React from 'react';
import { createRoot } from 'react-dom/client';
```

Por qué dos modulos para react?

DEPENDENCIAS

1. Importamos las dependencias

```
import React from 'react';
import { createRoot } from 'react-dom/client';
```

Core de React

Ignora la plataforma

Para la web

Interacciones con el DOM

- react-dom para la web
- react-native para mobile o desktop
- react-three-fiber para escenas 3D usando WebGL y Three.js

DEPENDENCIAS

Cada uno tiene sus primitivos

→ react-dom para la web

→ elementos del DOM: <p></p>, <h1></h1>, <div></div>

→ react-native para mobile o desktop

→ Text, View & Pressable

→ react-three-fiber para escenas 3D usando WebGL y Three.js

→ elementos 3D: luces, cámaras, geometrías, materiales

CREANDO UN ELEMENTO

2. Creamos un elemento de React

```
const element = React.createElement(  
  'h1' ,  
  { id: 'hola' } ,  
  'Hola React'  
)
```

React.createElement()

Función que acepta 3 parámetros

- Tipo de elemento que se quiere crear ('p', 'a', 'section')
- Propiedades del elemento (id, className)
- Contenido del elemento

CREANDO UN ELEMENTO

2. Creamos un elemento de React

```
const element = React.createElement(  
  'h1' ,  
  { id: 'hola' } ,  
  'Hola React'  
);  
  
console.log(element);
```



```
{  
  type: "p",  
  key: null,  
  ref: null,  
  props: {  
    id: 'hola',  
    children: 'Hola React',  
  },  
  _owner: null,  
  _store: { validated: false }  
}
```

RENDEREANDO

3. Rendereamos la aplicación

```
const container = document.querySelector('#root');
const root = createRoot(container);
root.render(element);
```

Este ya lo conocemos re-bien!
root va a ser el contenedor de nuestra aplicación

RENDEREANDO

3. Rendereamos la aplicación

```
const container = document.querySelector('#root');
const root = createRoot(container);
root.render(element);
```

createRoot()

Función de **react-dom**

Le indica a React que este va a ser el contenedor **Root** (raíz) de nuestra aplicación

RENDEREANDO

3. Rendereamos la aplicación

```
const container = document.querySelector('#root');
const root = createRoot(container);
root.render(element);
```



render()

función que convierte elementos React a nodos del DOM

RENDEREANDO

3. Rendereamos la aplicación

```
const container = document.querySelector('#root');
const root = createRoot(container);
root.render(element);
```

```
{
  type: "p",
  key: null,
  ref: null,
  props: {
    id: 'hola',
    children: 'Hola React',
  },
  _owner: null,
  _store: { validated: false }
}
```



```
<p id='hola'>
  Hola React
</p>
```

RENDEREANDO

3. Rendereamos la aplicación

```
const container = document.querySelector('#root');
const root = createRoot(container);
root.render(element);
```

ANTES DE LA VERSION 18

A partir de la versión 18

```
ReactDOM.render(
  element,
  container
);
```

CREANDO NUESTRO PROPIO REACT

EJERCICIO

Vamos a crear nuestra propia función render() con JS vainilla para entender como funciona react

La función debe de aceptar dos parámetros:

- El elemento que queremos crear (tipo & propiedades)
- El elemento contenedor al que le vamos a agregar el elemento creado

Criterios de aceptación:

- Se debe de mostrar en la página un elemento `<a>`, que diga “Leer más acerca de React”, y que nos lleve a la página <https://react.dev/>
- Debe de funcionar para cualquier etiqueta HTML
- Debe de funcionar con cualquier atributo HTML (`href`, `id`, `disabled`, ...)
- El elemento debe de contener el texto especificado en la llave `children`. `children` siempre va a ser un string

[Leer más acerca de React](#)

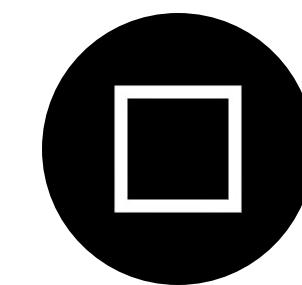
CREANDO NUESTRO PROPIO REACT

CÓDIGO INICIAL

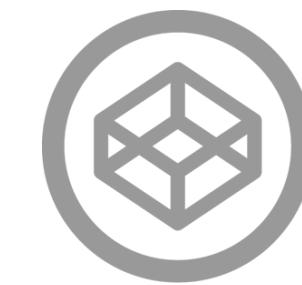
```
function render( element, container ) {
  /* tu código aquí */
}

const element = {
  type: 'a',
  props: {
    href: 'https://react.dev',
  },
  children: 'Leer más acerca de React',
}

const container = document.querySelector('#root');
render(element, container);
```



[sandbox](#)



[codepen](#)

Ya vimos la forma “complicada” de crear un elemento react con `React.createElement()`

```
const element = React.createElement(  
  'h1' ,  
  { id: 'hola' } ,  
  'Hola React'  
) ;
```

Pero es una sintaxis muy elaborada por lo que el estándar es **JSX**

```
const element = (  
  <p id='hola'>  
    Hola React  
  </p>  
) ;
```

POR QUÉ JSX?

```
const element = React.createElement(  
  'nav' ,  
  { id: 'main-nav' },  
  //children:  
  React.createElement(  
    'ul',  
    null,  
    React.createElement(  
      'li',  
      { id: 'nav-li' },  
      React.createElement(  
        'a',  
        { href: '/' },  
        'Inicio'  
      )  
    )  
  );
```

VS

```
const element = (  
  <nav id="main-nav">  
    <ul>  
      <li id="nav-li">  
        <a href="/">  
          Inicio  
        </a>  
      </li>  
    </ul>  
  </nav>  
);
```

POR QUÉ JSX?

- el navegador no entiende jsx
- necesitamos transpilar jsx a js
- esto se hace en el proceso de construcción o “build”
- se importa react pero no lo vemos 😊

ESPACIOS PARA “EXPRESIONES”

- en jsx, todo lo que ponemos entre una etiqueta html de apertura y una de cierre se trata como un string
- los espacios para expresiones se crean a través de llaves {}
- esto se hace en el proceso de construcción o “build”
- y solo acepta expresiones 🤔 (no declaraciones/statements)
- comentarios:

```
const element = (
  <nav id="main-nav">
    { /* un comentario */
  </nav>
);
```

ATRIBUTOS EN LOS ESPACIOS DE EXPRESIONES

Podemos usar estos espacios para agregar atributos dinámicos

```
const someId = 'my-id';

const element = (
  <main id={someId}>
    Hola React
  </main>
);
```

Atributos Booleanos



```
<input required />
```



```
<input required={true} />
```

HTML VS JSX

Ojo con las palabras reservadas, recordemos que estamos escribiendo dentro de un archivo JS

```
const element = (
  <label for='firstName' />
  Tu nombre:
  </label>
  <input id='firstName' class='text-input' >
);
```

HTML VS JSX

Qué hacemos con ese `for` y `class`

```
const element = (
  <label for='firstName' />
  Tu nombre:
  </label>
  <input id='firstName' class='text-input' >
);
```

HTML VS JSX

```
const element = (
  <label htmlFor='firstName'>
    Tu nombre:
  </label>
  <input id='firstName' className='text-input'>
);
```

HTML VS JSX

Otras reglas de JSX

→ html es muy relajado, no le importa si no cerramos etiquetas.
jsx nos muestra un error si no tenemos etiquetas de cierre.

CÓDIGO HTML VÁLIDO

```
<div>
  <p> Este es un párrafo
  <p> Este es otro párrafo
</div>
```

CÓDIGO JSX VÁLIDO

```
const element = (
  <div>
    <p> Este es un párrafo </p>
    <p> Este es otro párrafo </p>
  </div>
);
```

HTML VS JSX

Otras reglas de JSX

- html es muy relajado, no le importa si no cerramos etiquetas.
jsx nos muestra un error si no tenemos etiquetas de cierre.

CÓDIGO HTML VÁLIDO

```
<div>
  <img alt='paisaje de las montañas' src='./src/img/mountains.png'>
</div>
```

CÓDIGO JSX VÁLIDO

```
<div>
  <img alt='paisaje de las montañas' src='./src/img/mountains.png' />
</div>
```

HTML VS JSX

Otras reglas de JSX

- html no es case-sensitive (sensible a mayúsculas).
de hecho, una práctica muy común hace muchos años
era poner todas las etiquetas html en mayúsculas
- jsx es case-sensitive, todas las etiquetas html deben estar
escritas en minúsculas

HTML VS JSX

Otras reglas de JSX

→ html no es case-sensitive (sensible a mayúsculas).
de hecho, una práctica muy común hace muchos años
era poner todas las etiquetas html en mayúsculas

CÓDIGO HTML VÁLIDO

```
<DIV>
  <IMG alt='paisaje de las montañas' src='./src/img/mountains.png'>
</DIV>
```

HTML VS JSX

Otras reglas de JSX

- html no es case-sensitive (sensible a mayúsculas).
de hecho, una práctica muy común hace muchos años
era poner todas las etiquetas html en mayúsculas
- jsx es case-sensitive, todas las etiquetas html deben estar
escritas en minúsculas

```
const element = (
  <div>
    <p> Este es un párrafo </p>
    <p> Este es otro párrafo </p>
  </div>
);
```

HTML VS JSX

Otras reglas de JSX

- “traducción” de los atributos html (kebab-case) a jsx (camelcase)
 - ⚠ dos excepciones a esta regla son los **data-attribute** y **aria-attribute**

HTML VS JSX

Otras reglas de JSX

- estilos en línea
- jsx acepta un objeto dentro del atributo style para aplicar estilos en línea

```
const element = (
  <h1 style={{fontSize: '2rem'}}>
    Hola React!
  </h1>
);
```

- 🤔 doble llave? qué pasó ahí?

HTML VS JSX

Otras reglas de JSX

- estilos en línea
- 🤔 doble llave? qué pasó ahí?

```
const customStyle = {  
  fontSize: '2rem',  
  color: 'hotpink',  
};  
  
const element = (  
  <h1 style={customStyle} >  
    Hola React!  
  </h1>  
>);
```

BARRA DE BÚSQUEDA

Buscar:

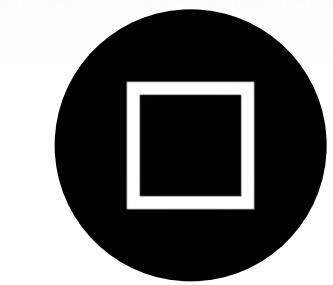


EJERCICIO

Vamos a convertir el código HTML a JSX. En el link de Sandbox tienes el HTML en el archivo ./src/App.js

Criterios de aceptación:

- El UI se debe de ver como en la imagen de arriba
- No deben de aparecer errores en la consola
- No deben aparecer alertas (warnings) en la consola



sandbox

REDISEÑO DE INSTAGRAM

EJERCICIO

Vamos a convertir el código HTML a JSX. En el link de Sandbox tienes el HTML en el archivo ./src/App.js

En el mismo archivo encontraras un objeto post, el cual contiene toda la información necesaria para llenar los datos

Criterios de aceptación:

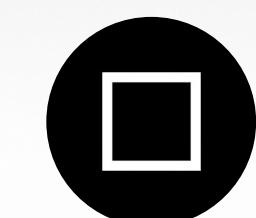
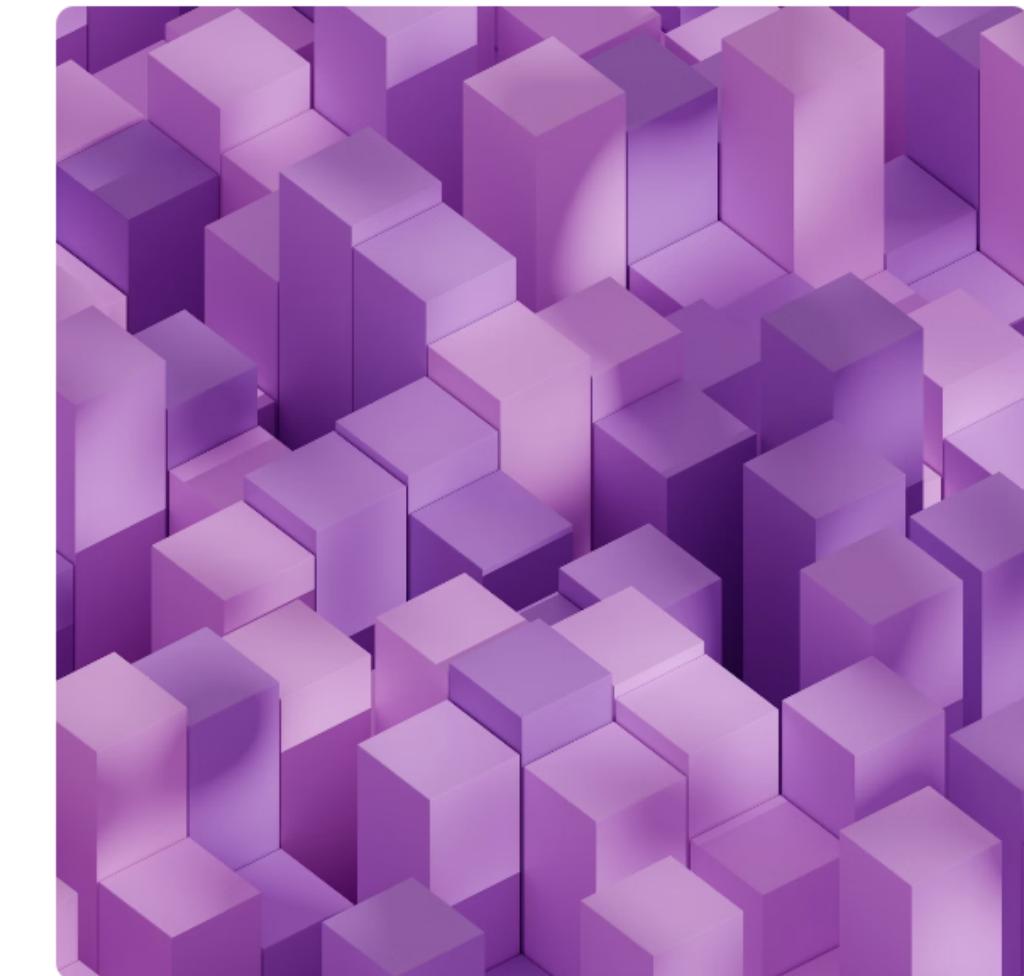
- El UI se debe de ver como en la imagen
- Los datos deben ser una referencia al objeto post usando expresiones.
No copiar y pegar
- El nombre del usuario debe de llevar a la página “/users/[handle]”,
en este caso “/users/helenmiles”
- La imagen de avatar debe de tener como texto alternativo “[name] Profile Picture”,
en este caso “Helen Miles Profile Picture”
- Para la fecha usa la función formatDate. Pasa como argumento el time stamp para obtener una fecha bien formateada



Helen Miles
Vancouver, CA

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Suspendisse
cursus facilisis congue. Mauris eget
pulvinar leo.

Posted March 14th at 08:09 AM



sandbox