

React JS

Introducción

React JS es una librería para construir interfaces web, es decir una librería de FrontEnd



Nos enfocamos a la interfaz del Front !

Creado por el equipo de desarrollo de Facebook e Instagram para solucionar el gran problema que tenían cuando su sistema se colapsaba debido al volumen tan enorme de información que manejan y la forma en que la manejaban.

El mayor problema se centraba en el Binding entre vistas y datos , es decir la forma en que FaceBook enlazaba los datos con la manera de presentarlos en su interfaz



Con la creación de React Facebook vió crecer en eficiencia y rendimiento todos sus sistemas.

Para meternos un poco en el contexto vamos a decir que la web está construida en base a 3 conceptos básicos:

- **HTML 5:** Es la estructura , la semántica , la información de la página web

- **CSS:** Es el maquillaje para la página , que la información se vea bonita, que se adapte a todos los tamaños de pantalla, etc...
- **Java Script:** Ya es programación, la página ya está viva , responde a las acciones del usuario, dicho de otro modo reacciona a sus acciones.

Siempre hemos aprendido que estos 3 conceptos deben de estar separados

Estructura(HTML) , presentación(CSS) y comportamiento (JS)

Y poníamos las 3 separadas en carpetas,asi son las buenas prácticas.



Pero llegó el equipo de React y dijo que separar en HTML , CSS y Java Script no era la mejor idea, la mejor idea es meterlos en un solo paquete llamado **componente**



Pero esto en teoría va contra las buenas prácticas no? Si queremos escalar nuestro proyecto y resulta que tenemos todo junto metido en un paquete cuando queramos ampliar nuestro código va a parecer que volvemos a los años 90 cuando en el propio HTML metíamos atributos style, metíamos llamadas a eventos también en el propio HTML, etc.. ,no vamos a poder desacoplar el código, ni crecer en nuevas funcionalidades, etc...

Por qué entonces proponen esta idea?

Pues bien, ellos dicen que cuando tu separas tu proyecto en las 3 tecnologías HTML, CSS y JS , realmente estás separando **por código** y no **por funcionalidad**.

Una interfaz web estaría constituida por ejemplo por un logo, el carrito de la compra , un icono de búsqueda, un par de botones , etc...es decir está constituida por 'piezas'

La filosofía de React es separar el proyecto por piezas de interfaz en vez de por código.

De este modo estas piezas o componentes son reutilizables. Si tengo 2 botones y tengo que crear un tercer botón no tengo que escribir de nuevo el código del botón sino que puedo reutilizar uno de los componentes botón que ya tengo creados.

Para React JS la estructura (HTML) y la lógica o comportamiento (JS) son inseparables.



Si se puede desacoplar el código y la funcionalidad , pero ahora ya no va a estar dividido en HTML, CSS y JS sino por funcionalidad en la interfaz, por piezas por bloques de interfaz.

Imaginemos que hemos desarrollado un menú de navegación en un proyecto y queremos reutilizarlo en otro proyecto.

Por qué tengo que volver escribir el código del menú en el nuevo proyecto?

Y menudo lío cuando escriba el HTML y quiera traerme el java script , de donde me lo traigo? Cual debo traerme? Que funciones cojo? Y si alguna función está comprometida con otra función que recibe un determinado parámetro y resulta que cuando se ejecute no lo recibe bien?

Donde están los estilos?

Es un caos!

La estructura y la lógica por lo tanto están unidos de forma inseparable en el componente.

El hecho es que la estructura y el comportamiento deben de ir unidos, no sucede así con el CSS que tendremos la opción de meterlo también dentro del componente o bien dejarlo fuera .Este es hoy en día uno de los debates más calientes del java script moderno y de momento no hay ninguna razón de peso aplastante que se incline por una u otra.

Que es un componente



Los componentes son las piezas de la interfaz (UI), las piezas con las que el usuario interactúa, en definitiva lo que ves en la pantalla.

Pueden reutilizarse y combinarse entre ellos para formar otros componentes más complejos que a su vez se combinan y forman componentes mayores.

Para lograr esa unión entre el HTML y el JS React nos trae su ingrediente secreto, que se llama **JSX** (Java Script Extended o Java Script XML) que consiste en poder escribir código HTML dentro de java script.

Por debajo en el background de React actúan WebPack y Babel convirtiendo ese código a código Java Script compatible con los navegadores.

Vamos a ver un pequeño ejemplo:

Este código que vemos es un componente

```

JS import React from 'react' > [🔗] Button
1  import React from 'react'
2  const Button={()=>{
3    <a href="https://www.geekshubs.com"
4    class="button default" >
5    Información de Cursos Full Stack
6    </a>
7  export default Button
8

```

En la línea 1 observamos que importamos la librería de React para poder trabajar con todos sus métodos.

En la línea 10 exportamos el botón que en definitiva es nuestro componente , el cual vamos a reutilizar probablemente en otro lugar de nuestro proyecto o inclusive en otro proyecto.

De la línea 2 a la línea 6 es el código de nuestro componente que no es ni más ni menos que una simple función java script que está retornando HTML.

¡ Esto es JSX ! Con JSX devuelves contenido HTML.

Puedes pasar de HTML a React sin ser un pro en java script.

Virtual DOM

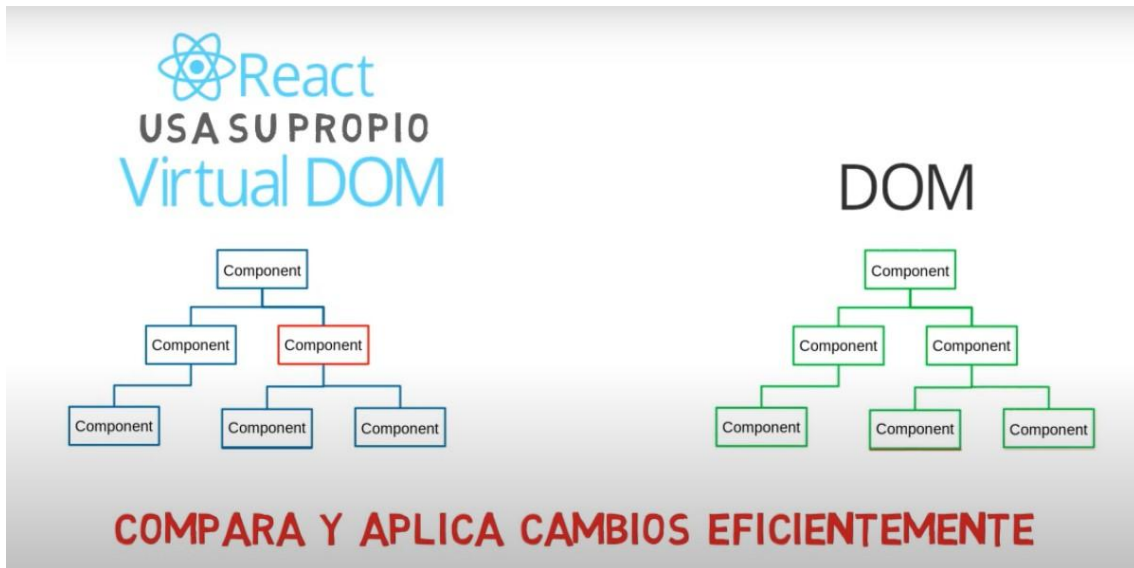
Otra cosa nueva que nos trae React es el Virtual DOM,.

El virtual Dom es una representación en memoria del DOM

El DOM es toda la estructura HTML del sitio, cuando tu cargas una página web el navegador lee el código HTML, lee el código CSS , lee el código Java Script , interpreta todo ese código, lo parsea incluso interpreta la geometría (en que lugar debe estar cada elemento) y lo muestra en pantalla, lo pinta.

Cuando hay un cambio en el DOM por ejemplo java script hace un cambio el navegador tiene que hacer ese proceso nuevamente esto es muy costoso en términos de recursos memoria RAM, procesador, etc...

Lo que hace React es guardar una copia en memoria del DOM y ahí va trabajando.



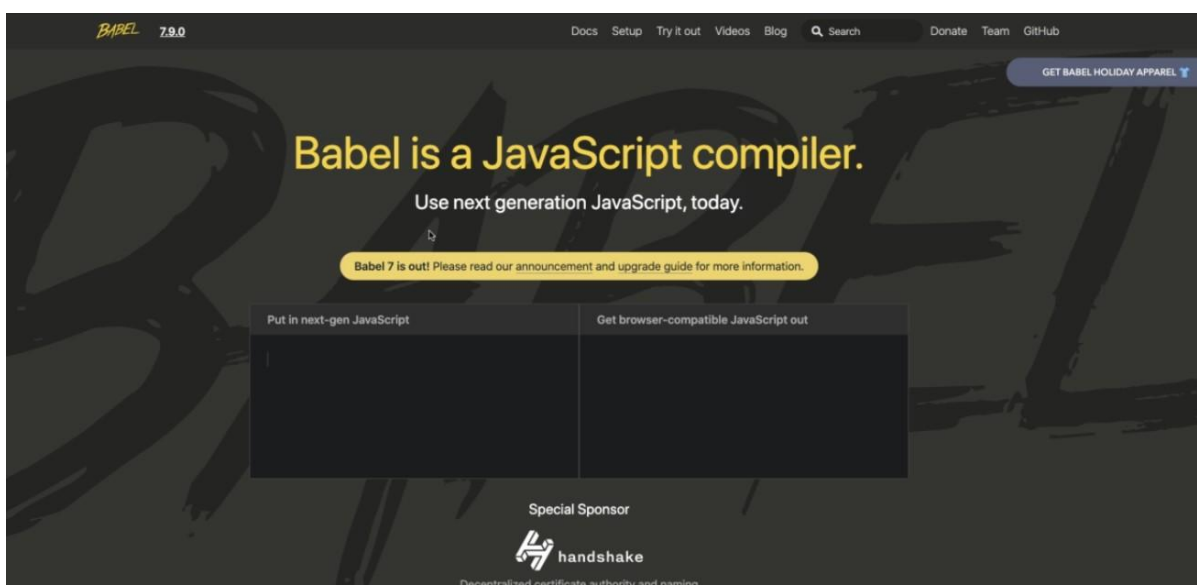
Que es Babel

Babel es un entorno que funciona en el Background cuando creamos las aplicaciones de React. Este entorno para nosotros es transparente pero resulta algo muy útil si entendemos bien el concepto de Babel.

Babel convierte código java script moderno en código compatible con todos los navegadores.

Nos permite usar características actuales de JS aceptadas en el último standard de ECMA Script y poder usarlas en navegadores más antiguos que aún no soportan esa característica sin perder la compatibilidad entre navegadores, como por ejemplo las 'templates string'

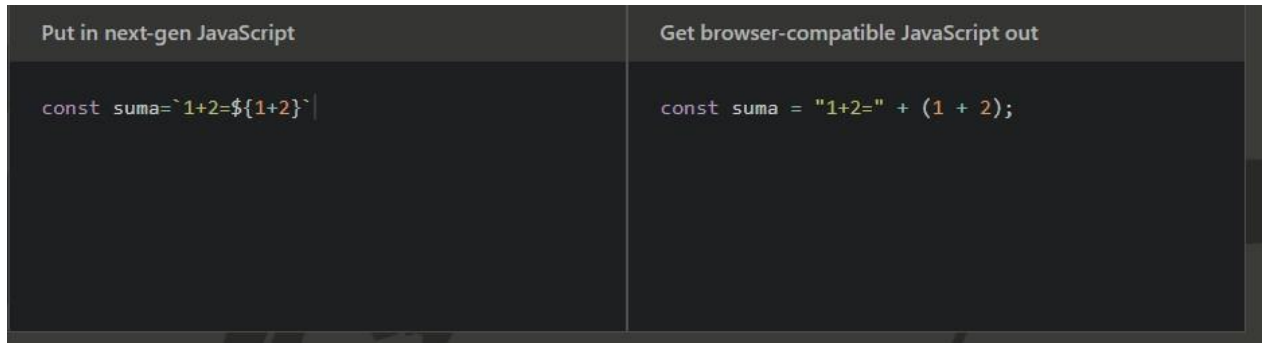
Podemos probar Babel en la página babeljs.io



En la parte izquierda del recuadro podemos escribir una instrucción JS moderna como por ejemplo una template string de la siguiente forma:

```
const suma=`1+2=${1+2}`
```

esta instrucción que vamos a colocar en la parte izquierda de las dos cajas que aparecen en la página web de babel, va a ser convertida automáticamente a javascript compatible con los navegadores en la caja derecha de la siguiente forma:



Instalaciones para REACT

Instalaciones recomendadas

🔗 Instalaciones Necesarias

- [Google Chrome](#)
- [React Developer Tools](#)
- [Redux Devtools](#)
- [Visual Studio Code](#)
- [Postman](#)
- [Mongo Compass](#)
- [Git](#)
- [Node](#)

Postman es un entorno para realizar pruebas a Apis y también lo usaremos para probar nuestro propio Backend.

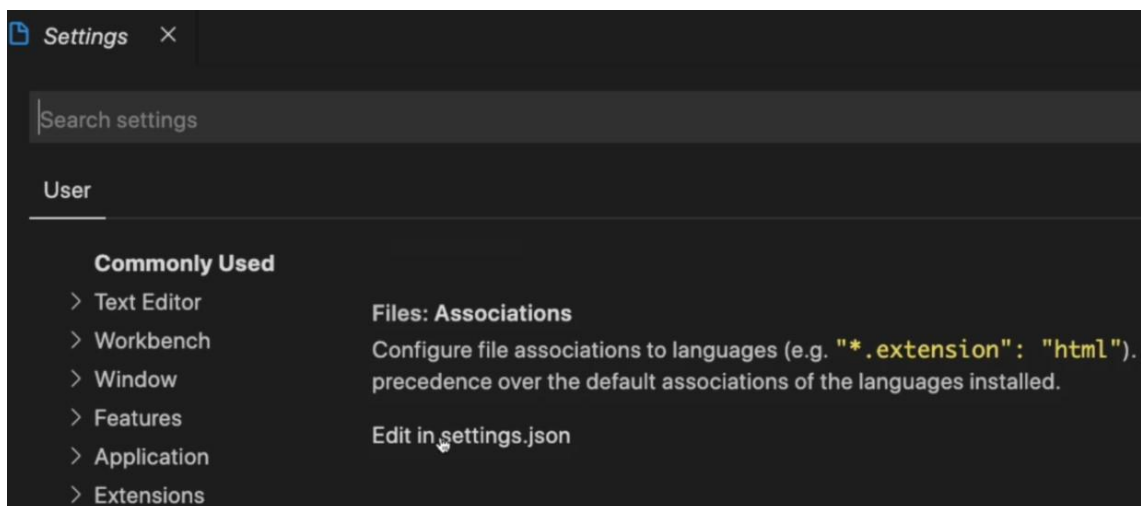
Mongo Compass nos va a permitir poder trabajar con la Bd Mongo en forma local aunque esté configurada en un servidor remoto.

Extensiones de VSCode

Activitus Bar

🔗 Configuración del Bracket Pair Colorizer 2

Bracket Pair Colorizer 2



```
"workbench.colorTheme": "Monokai Night",  
"bracket-pair-colorizer-2.colors": [  
  "#fafafa",  
  "#9F51B6",  
  "#F7C244",  
  "#F07850",  
  "#9CDD29",  
  "#C497D4"  
],
```


Código a pegar:

```
"bracket-pair-colorizer-2.colors": [  
  "#fafafa",  
  "#9F51B6",  
  "#F7C244",  
  "#F07850",  
  "#9CDD29",  
  "#C497D4"  
],
```

Snippets de React (Nos va a permitir trabajar más rápido con React)

Instalaciones recomendadas sobre React

- [ES7 React/Redux](#)
- [Simple React Snippets](#)
- [Auto Close Tag](#)

Primeros pasos con React:

¿Qué veremos en esta sección?

- Nuestra primera aplicación - Hola Mundo
- Exposiciones sobre los componentes
- Creación de componentes (Functional Components)
- Propiedades - Props
- Impresiones en el HTML
- PropTypes
- DefaultProps
- Introducción general a los Hooks

- useState

Es una sección importante, especialmente para todos los que están empezando de cero en React, ya que dará las bases de cómo segmentar la lógica de nuestra aplicación en pequeñas piezas más fáciles de mantener.



¿Qué es un componente React?

**Pequeña pieza de código encapsulada re-utilizable
que puede tener estado o no.**

Un componente es una pequeña pieza de código encapsulada que realiza un trabajo en específico.

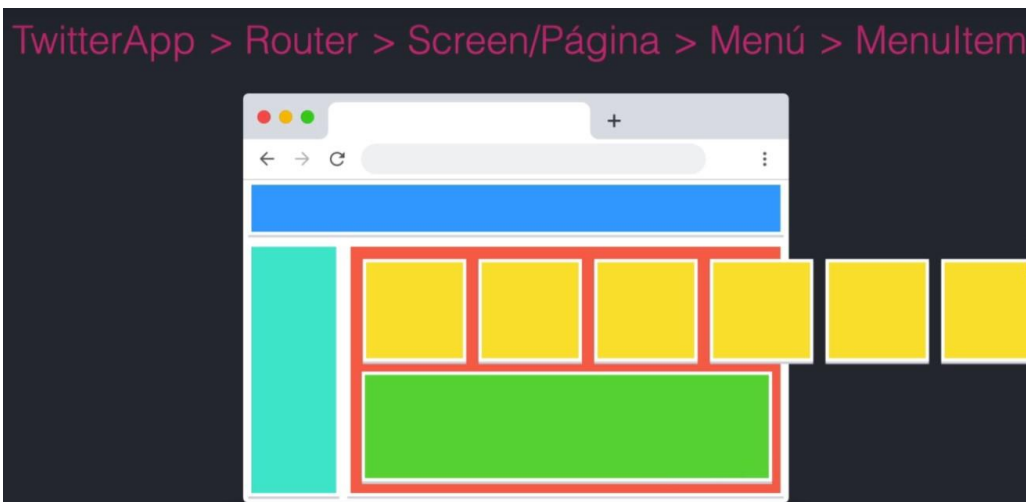
El estado es como se encuentra la información del componente en un punto determinado del tiempo

Esta sería una posible subdivisión en componentes de la interfaz web de twitter:



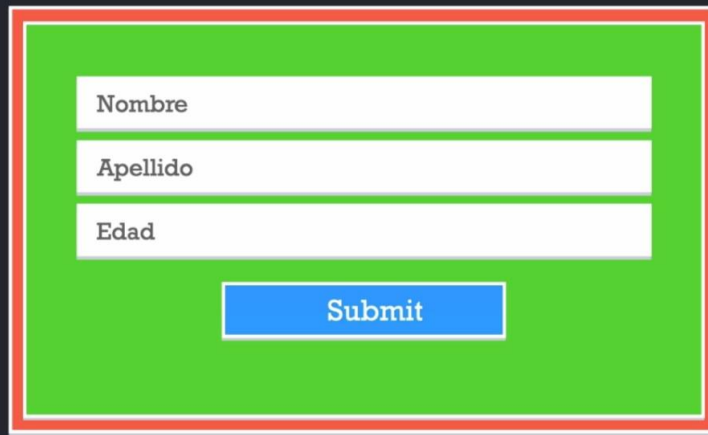
TwitterApp

TwitterApp > Router > Screen/Página > Menú > Menuitem



Cada uno de los componentes o piezas de color amarillo como se ve se repiten, serian la misma estructura pero podrían recibir argumentos diferentes , lo que les haría tener quizás una diferente funcionalidad. El recuadro verde podría ser un formulario de datos.

Formulario



Nombre

Apellido

Edad

Submit

Un formulario (componente) puede tener un estado inicial con sus datos en blanco , sin información, y en otro momento puntual en el tiempo podrá tener otro estado(cambia el estado) diferente tras haber rellenado los campos de texto con una determinada información.

Primera aplicación de React

Vamos a crear nuestro proyecto react desde la consola.

Utilizaremos nuestra carpeta react.

Nos situamos en esta carpeta desde la consola y teclearemos lo siguiente:

```
C:/react> npx create-react-app counter-app
```

counter-app será el nombre de nuestra aplicación react.

Pulsaremos enter y comenzará la instalación la cual puede demorarse unos minutos, depende de la velocidad de tu internet.

```
ca: npm
Microsoft Windows [Versión 10.0.18363.1016]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\dbeng>cd..

C:\Users>cd..

C:\>cd react

C:\REACT>npx create-react-app counter-app
npx: installed 98 in 20.238s

Creating a new React app in C:\REACT\counter-app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[■.....] | fetchMetadata: sill resolveWithNewModule scheduler@0.19.1 checking installable status
```

Una vez finalizado nos aparecerá esta pantalla:

```
ca: Símbolo del sistema

found 0 vulnerabilities

Created git commit.

Success! Created counter-app at C:\REACT\counter-app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

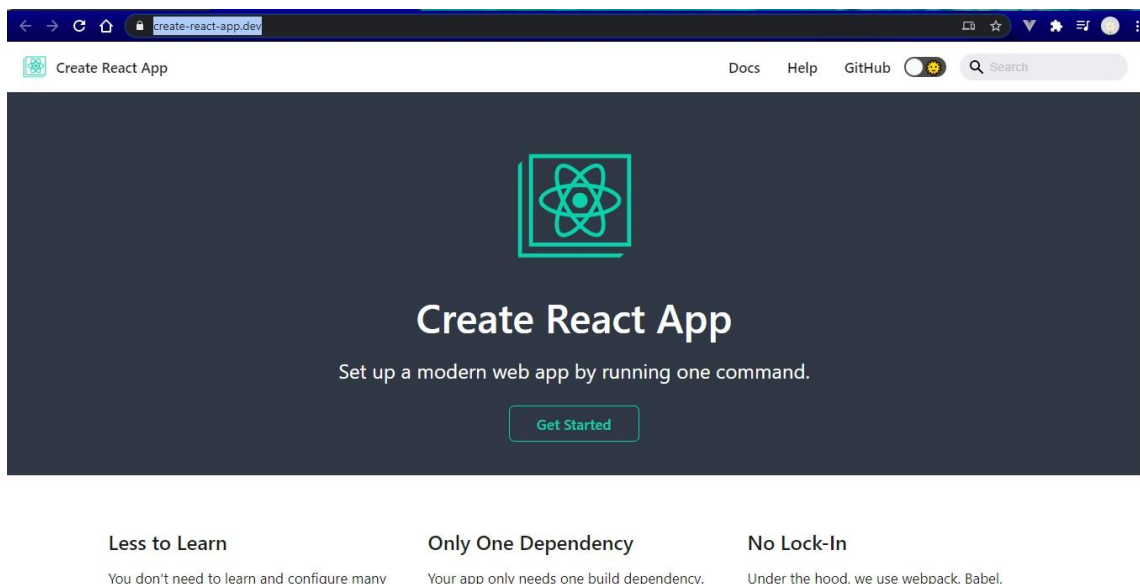
  cd counter-app
  npm start

Happy hacking!

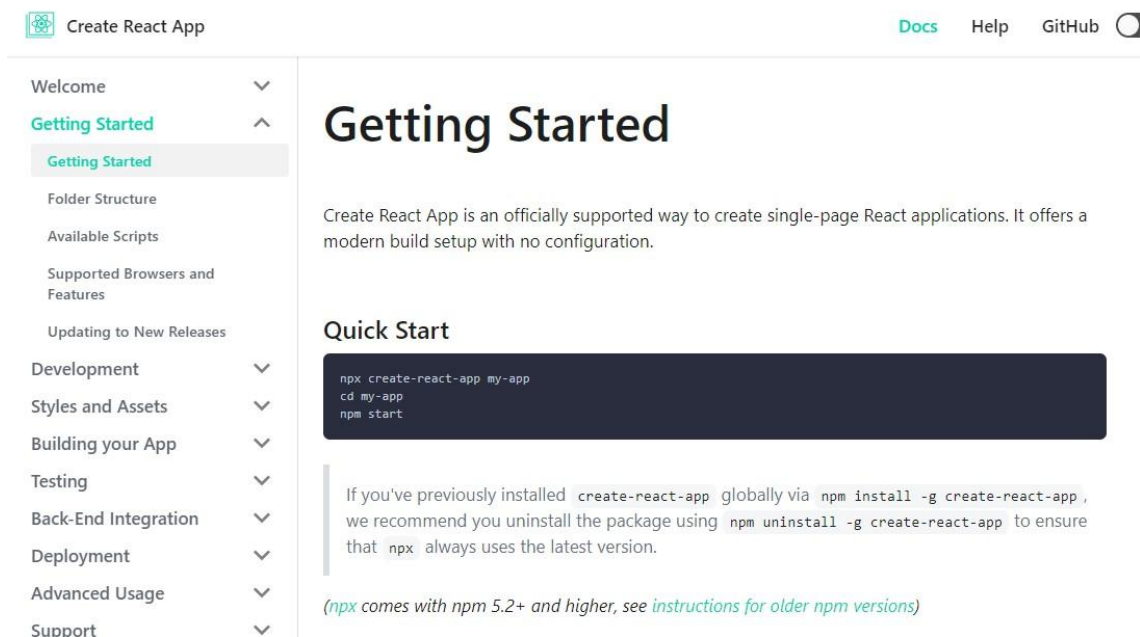
C:\REACT>
```

En esta dirección podemos acceder a la documentación oficial de apoyo de **create-react-app** por si necesitamos en un futuro consultar cualquier funcionalidad sobre este comando:

<https://create-react-app.dev/>

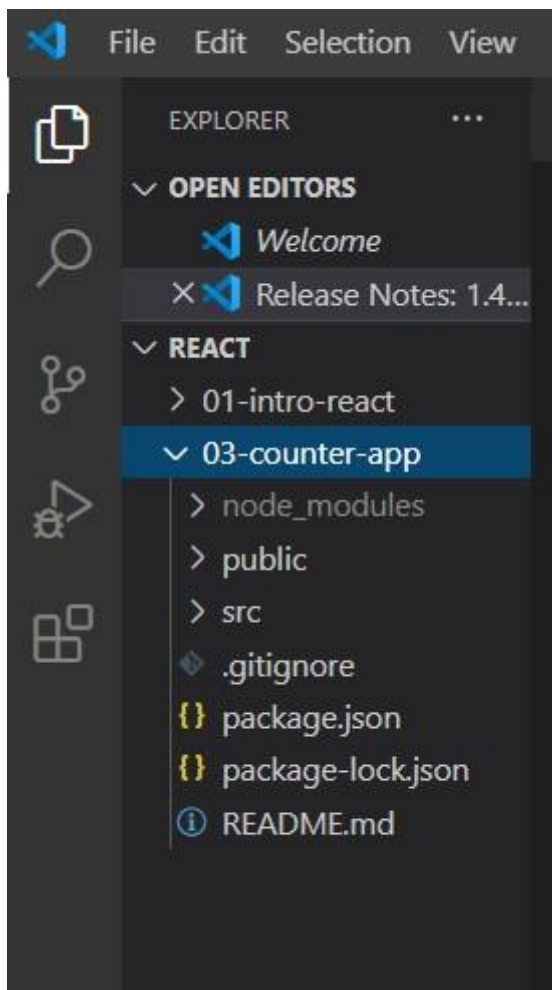


Ahora vamos a pulsar el botón de Get Started:



En la parte izquierda tenemos una ayuda dividida por temas que nos va a resultar muy útil.

Vamos a Visual Studio y abrimos nuestra carpeta React, veremos que nos ha creado la carpeta counter-app , la vamos a renombrar como 03-counter-app, vemos que dentro tiene la siguiente estructura:



Ahora abrimos de nuevo la consola y vamos a teclear lo siguiente , siempre desde nuestra localización en la carpeta React:

```
Windows PowerShell

C:\REACT>cd 03-counter-app

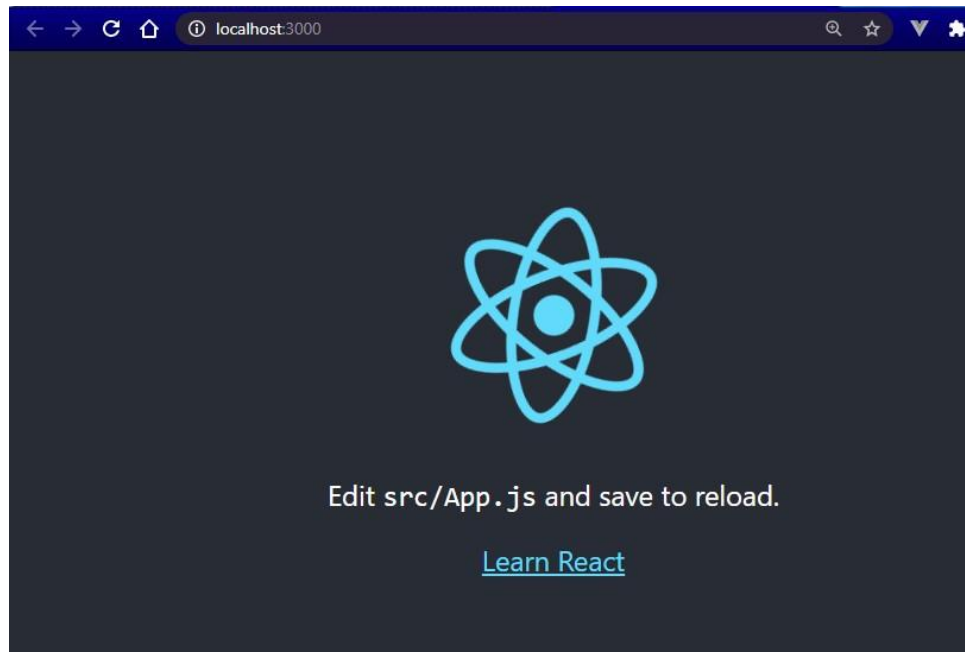
C:\REACT\03-counter-app>npm start

> counter-app@0.1.0 start C:\REACT\03-counter-app
> react-scripts start

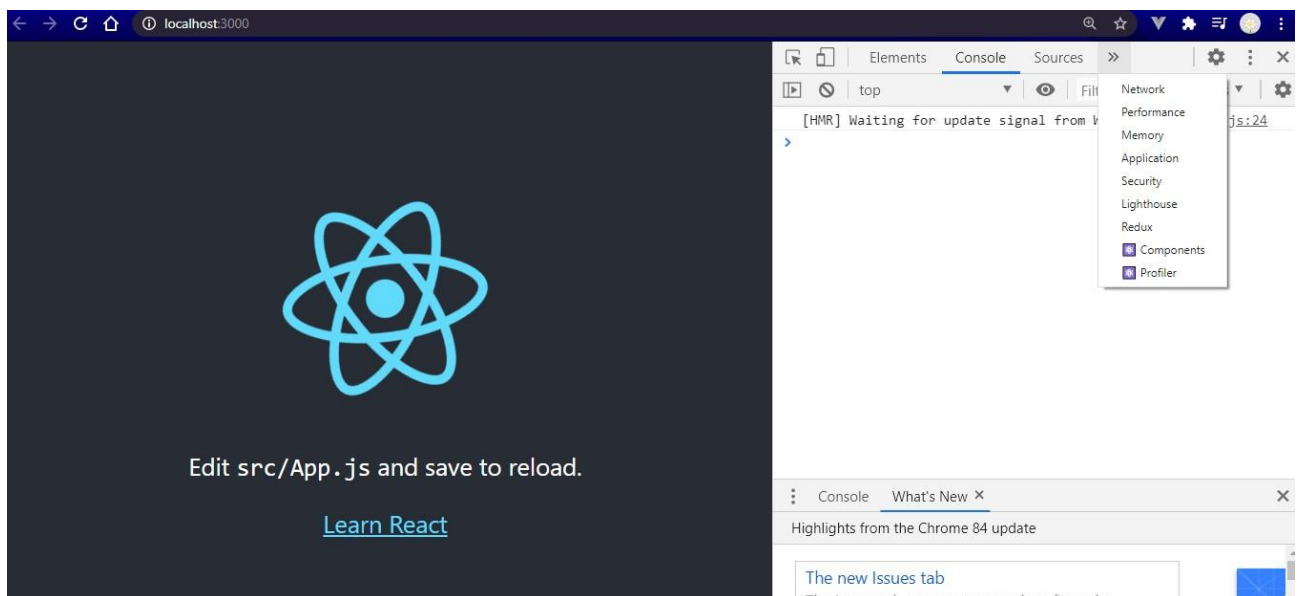
i @wds: Project is running at http://192.168.1.57/
i @wds: webpack output is served from
i @wds: Content not from webpack is served from C:\REACT\03-counter-app\public
i @wds: 404s will fallback to /
Starting the development server...
Compiled successfully!
```

cd 03-counter-app y después **npm start**

Si todo va bien nos arrancará nuestra aplicación React en localhost:3000



Con CTRL +Shift + i abrimos la consola y deberemos tener las pestañas de las herramientas de React que hemos instalado, por ejemplo Redux , si no nos aparecen deberemos instalarlas.



Estructura de directorios

node-modules:

Contiene librerías que utiliza react pero en desarrollo ,lo cual significa que cuando despluguemos una aplicación react no será necesario exportarlas.

Estas carpetas nosotros no las vamos a modificar nunca manualmente.

public:

Los iconos que aparecen logo192.png y logo512.png son por si nosotros instalamos nuestra aplicación web como una PWA (aplicación web progresiva)

Si abrimos el **manifest.json** los veremos en él la referencia a los iconos:

```
{ } manifest.json X
03-counter-app > public > { } manifest.json > ...
1  [
2    "short_name": "React App",
3    "name": "Create React App Sample",
4    "icons": [
5      {
6        "src": "favicon.ico",
7        "sizes": "64x64 32x32 24x24 16x16",
8        "type": "image/x-icon"
9      },
10     {
11       "src": "logo192.png",
12       "type": "image/png",
13       "sizes": "192x192"
14     },
15     {
16       "src": "logo512.png",
17       "type": "image/png",
18       "sizes": "512x512"
19     }
20   ],
```

Esto no tiene nada que ver React, React no maneja esta parte de las PWA.

Vamos a ver una comparativa para poder entender bien la diferencia entre una aplicación web, una aplicación nativa y una aplicación web progresiva.

Qué es una aplicación web?



Twitter es un ejemplo de aplicación web

Igual que existen las aplicaciones para Windows, Mac, Android o iOS, también existen otro tipo de aplicaciones que no dependen de ningún sistema operativo, sino que **toman lugar en una página web** en un navegador.

Las páginas web de Twitter y Facebook son dos buenos ejemplos de aplicaciones web. En ellas puedes **hacer prácticamente lo mismo que en las aplicaciones nativas para Android o iOS**, pero sin necesidad de instalar nada: solo necesitas un navegador con conexión a Internet.

La definición es suficientemente ambigua para incluir otros ejemplos que **no tienen por qué ser tan complejos** como las webs de Twitter o Facebook. Por ejemplo, podrían considerarse aplicaciones web un sitio para convertir divisas y otro para consultar el tiempo, aunque habrá quien difiera y opine que son simplemente páginas web sin más.

¿Qué es una aplicación nativa?

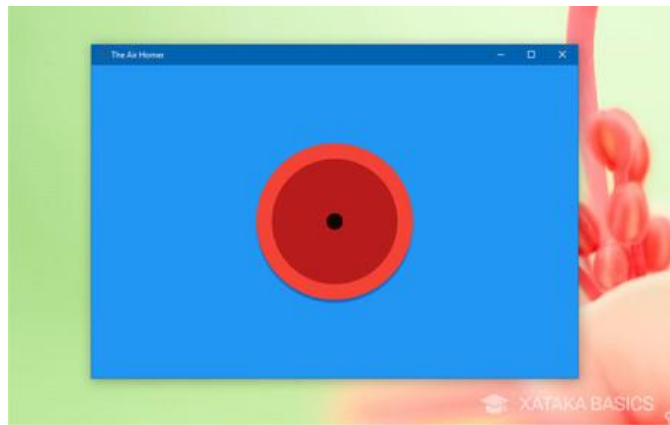


Aplicación nativa de Twitter para Windows 10

Probablemente hayas oído hablar el término de aplicación nativa, ¿pero qué es exactamente? En términos generales, es una aplicación que ha sido **específicamente desarrollada para el sistema operativo en el que corre**.

A diferencia de una aplicación web que funciona en distintos sistemas operativos como Windows, Mac o sistemas de móviles, una aplicación nativa ha sido desarrollada específicamente para un sistema operativo y, presumiblemente, **respeto mejor el aspecto y funcionamiento en dicha plataforma**, además de funcionar más fluida.

¿Qué es una aplicación web progresiva?



Una aplicación web progresiva sencilla en Windows 10 (vía Chrome)

Las aplicaciones web progresivas **están a medio camino entre las dos anteriores**: son básicamente páginas web, pero mediante el uso de *Service Workers* y otras tecnologías se comportan más como aplicaciones normales que como aplicaciones web.

Mediante los *Service Workers* y otras tecnologías las aplicaciones web progresivas pueden seguir **ejecutándose en segundo plano** sin tener que vivir dentro del navegador. En el móvil es posible instalarlas como una aplicación más y también en Windows mediante la mediación de Google Chrome y Mozilla Firefox. **Windows 10** va a añadir soporte para aplicaciones web progresivas en la tienda de Microsoft en la próxima gran actualización, con nombre en clave Redstone 4.

En resumen, las aplicaciones web progresivas son una evolución natural de las aplicaciones web que **difuminan la barrera entre la web y las aplicaciones**, pudiendo realizar tareas que generalmente solo las aplicaciones nativas podían llevar a cabo. Algunos ejemplos son las notificaciones, el funcionamiento sin conexión a Internet o la posibilidad de probar una versión más ligera antes de bajarte una aplicación nativa de verdad.

El archivo robots tampoco tiene que ver mucho con React. El único archivo importante va a ser el **index.html**

Toda esta parte es para las aplicaciones PWA

```
<link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
<!--
  manifest.json provides metadata used when your web app is installed
  on a
  user's mobile device or desktop. See https://developers.google.com/
web/fundamentals/web-app-manifest/
-->
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
<!--
  Notice the use of %PUBLIC_URL% in the tags above.
  It will be replaced with the URL of the `public` folder during the
  build.
  Only files inside the `public` folder can be referenced from the HT
  ML.

  Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico"
  will
  work correctly both with client-side routing and a non-
  root public URL.
  Learn how to configure a non-
  root public URL by running `npm run build`.
-->
```

Archivo **gitignore**

Está en formato en json y es muy delicado.

Normalmente nosotros no vamos a hacer ninguna modificación en este archivo.

Es usado por git para decirle que archivos y carpetas quiero que ignore y no le de un seguimiento

Esta parte es muy importante

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
```

```
"eject": "react-scripts eject"
},
```

Aquí tenemos una descripción de los 4 Scripts:

<https://create-react-app.dev/docs/available-scripts/>

Cuando ejecutamos el comando npm start ya ejecutamos ese script

```
"start": "react-scripts start",
```

Esto le dice a node que busque el paquete react-script que contiene un conjunto de instrucciones y ejecute el start , esto arranca el servidor , configurar babel, escuchar cambios, etc...

Por defecto está lanzando el index.

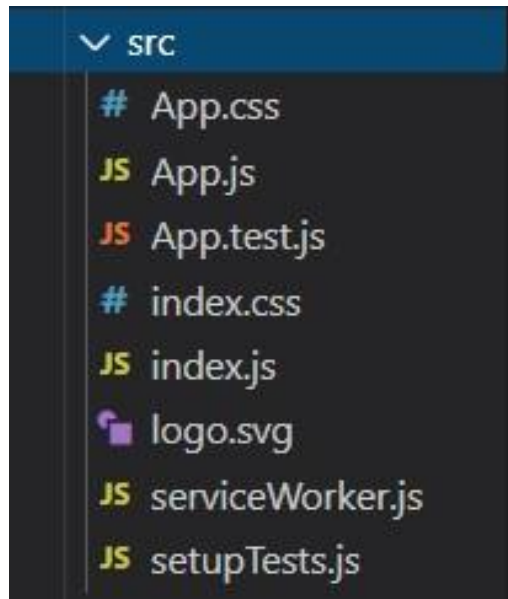
Test nos va a ayudar a que la aplicación tenga el menor número de errores posibles.

Con el eject vamos a poder cambiar esa configuración , pero no lo vamos a ejecutar porque si no a partir de ahí vamos a tener que hacer todo manualmente. No vamos a poder tener la aplicación como la tenemos en este momento.

Readme.md(md es de mark down) .No es más que un archivo que describe la aplicación.

package-lock.json.Define todas las versiones de los paquetes utilizados en la aplicación. Este archivo no se puede tocar.

Carpeta SRC



App.css

Es un archivo de estilos por defecto. Afecta al componente APP

App.js

Contiene el componente App.aquí importa el archivo css anterior

App.test.js

Lo importante es la parte final .test.js es , un archivo para realizar pruebas con el comando test

index.css

Archivo de estilos global de la aplicación, está importado en el index.js

index.js

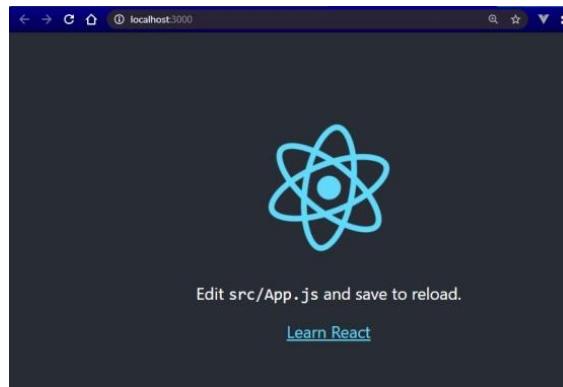
Es el punto inicial de la aplicación, cuando se lanza el webpack se lanza este archivo

serviceWorker.js

Propiamente de la PWA , no tiene que ver con React.Más bien tiene que ver con aplicaciones web progresivas.

logo.svg

El logo que aparece de React girando



setupTests.js

Se ejecuta exactamente en el momento en que nosotros levantamos por primera vez la parte de las pruebas

`npm run test` ejecuta este archivo primero para configurar todo lo necesario para las pruebas.

Borraremos todo, para empezar desde cero y ver como se configura todo desde cero.

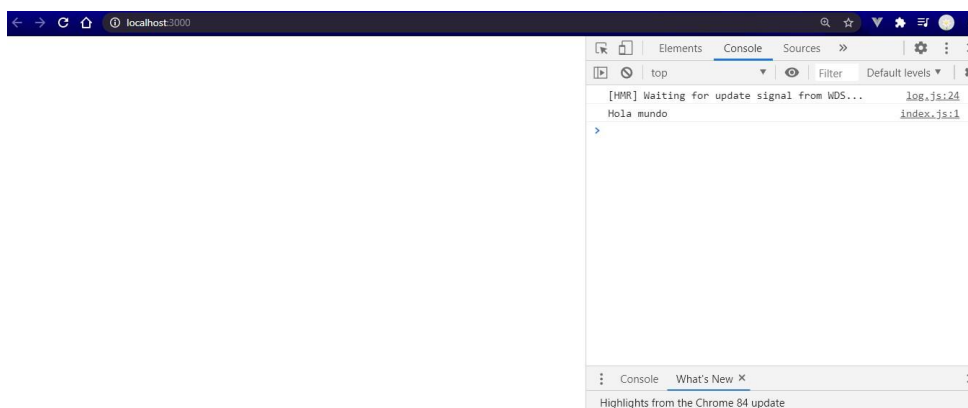
Hola mundo

Vamos a borrar todo el contenido de la carpeta **src**.

Creamos el archivo `index.js` y dentro ponemos:

```
console.log('Hola mundo')
```

Ahora al grabar, nos vamos a la página principal y vemos que el contenido ha desaparecido y en la consola aparece la salida del `console.log` (Hola Mundo).



Recordemos que para poder ver estos cambios tendremos que haber ejecutado **npm start**, que lo que hace es levantar nuestro servidor **webpack** de desarrollo para que estemos viendo todos estos cambios, genere babel, etc...

index.js

```
import React from 'react'
import ReactDOM from 'react-dom'

const saludo=<h1>Hola Mundo</h1>;
const divRoot=document.querySelector('#root');

ReactDOM.render(saludo,divRoot);
```

Nuestro primer componente React

La incorporación de los hooks en React nos permite desarrollar **componentes funcionales con estado y ciclo de vida** y por lo tanto, prescindir de los componentes de clase.

En Octubre de 2018, en la **React Conf**, el equipo de Facebook encargado del desarrollo del proyecto React (en ese momento Sophie Alpert y Dan Abramov) presentó a la comunidad una nueva propuesta: los **Hooks**. En Febrero de 2019, la versión 16.8 de React fue liberada al público y con ella esta nueva característica que introdujo uno de los cambios más interesantes en esta librería de desarrollo de interfaces de usuario. En React Native, los hooks fueron incorporados a partir de la versión 0.59.

¿Qué problemas tenía React y vienen a solucionar los Hooks?

Tras cinco años de existencia de la librería y gracias a la experiencia acumulada en el desarrollo de componentes, el equipo de React había logrado identificar tres problemas que debía corregir:

En primer lugar, la posibilidad de reutilizar lógica entre componentes. Los dos patrones principales para compartir código entre componentes, hasta ese momento, eran los *High-Order Components* y las *Render Props*. Si bien son formas correctas para lograr este objetivo, en algunos casos complejos esto podría derivar en un árbol de componentes anidados tan extenso que se conoce como *Wrapper Hell*.

Otro problema habitual es el desarrollo de componentes complejos y extensos.

Finalmente, las clases de Javascript suponen una dificultad no sólo para desarrolladores, sino para máquinas también. Por poner un ejemplo, es común la dificultad para entender la utilización de binding para no perder el contexto cuando queremos referenciar un método con la palabra reservada `this`. En el caso de las máquinas también se identificaron dificultades para implementar *hot-reloading* de manera fiable por el uso de clases, y algunos problemas para mejorar la *performance* de los componentes debido a patrones que dificultan la optimización en el momento de la compilación.

La solución: encontrar una alternativa a los componentes de clase

Si bien existía la posibilidad de intentar solucionar estos problemas de manera individual, es probable que la solución de uno de ellos implicase el agravamiento de los otros dos. El equipo de React entendió que estos no eran tres problemas aislados, sino tres síntomas de un mismo problema: la falta de un componente más simple que las clases, capaz de tener estado y ciclo de vida.

Para ello surgen los **hooks**, que son funciones que permiten enganchar nuestros componentes funcionales a características propias de un componente de clase. Es decir, proporcionan un estado y un ciclo de vida a estos componentes evitándonos a los desarrolladores el uso de las clases.

Una vez hecha esta introducción vamos a empezar a desarrollar nuestro primer componente.

Vamos a crear dentro de **src** un archivo llamado `PrimeraApp.js`.

Vamos a utilizar la nomenclatura upper camel case (capitalizamos la palabra es decir la primera letra de cada palabra en mayúscula, así podremos en el futuro reconocer a primera vista un componente).





Como dijimos anteriormente hay dos tipos de componentes los basados en clases y los basados en funciones. Nosotros vamos a trabajar con los componentes basados en funciones denominados **functional components**.

Antes se llamaban STF stateless functional components , pero cuando se introdujeron los hooks ya se denominan FC porque ya son capaces de manejar el estado.

Atajo: imr + TAB ya importa React en el código.

Código de nuestro primer componente(PrimeraApp.js):

```
import React from 'react'
//FC
const PrimeraApp = () => {
  return <h1>Hola mundo</h1>;
}
export default PrimeraApp;
```

Código del index.js:

```
import React from 'react'
import ReactDOM from 'react-dom'
import PrimeraApp from './PrimeraApp'
//const saludo=<h1>Hola Mundo</h1>;
const divRoot=document.querySelector('#root');

ReactDOM.render(<PrimeraApp />,divRoot);

//console.log('Hola mundo')
```

Ahora vamos a crear un estilo dentro de src

Creamos el archivo index.css

Conviene que las instrucciones css estén ordenadas alfabéticamente.

```
html,body{
  background-color: #21232A;
  color:white;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 1.3rem;
  padding: 70px;
}
```

Y dentro de index.js tendremos que importarlo para que pueda ser mostrado.

```
import React from 'react'
import ReactDOM from 'react-dom'
import PrimeraApp from './PrimeraApp'
import './index.css';
```

Fragment

Tenemos que tener algunas consideraciones teniendo en cuenta que nuestro código pasa por Babel en el background.

Por ejemplo si realizamos esta pequeña modificación en el código:

```
import React from 'react'
//FC
const PrimeraApp = () => {
  return
    <h1>Hola mundo</h1>;
}
export default PrimeraApp;
```

Hemos metido un salto de línea después del return, y aparentemente es lo mismo pero no lo es., se interpreta como que después del return hubiese un ;
return ;

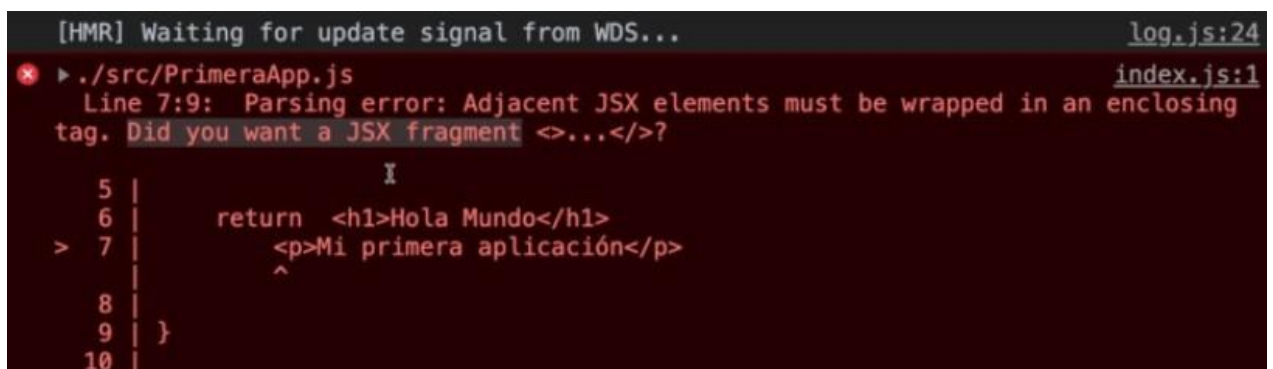
Y esto nos saca del script con lo que no se ejecutaría lo que viene detrás.

```
import React from 'react'
//FC
```

```
const PrimeraApp = () => {
  return
    <h1>Hola mundo</h1>;
    <p> Mi primera aplicación </p>
}
export default PrimeraApp;
```

Una función js no puede devolver dos elementos a la vez (esto es js no de react).

Este es el error que se produce:



```
[HMR] Waiting for update signal from WDS... log.js:24
✖ ./src/PrimeraApp.js index.js:1
  Line 7:9:  Parsing error: Adjacent JSX elements must be wrapped in an enclosing
tag. Did you want a JSX fragment <>...</>?

    5 |         |
    6 |         |   return <h1>Hola Mundo</h1>
  >  7 |         |     <p>Mi primera aplicación</p>
      |         |     ^
    8 |         |
    9 |         |   }
   10 |         |
```

De alguna manera tenemos que lograr que se devuelvan los 2 elementos a la vez. Podemos devolver un primitivo o un arreglo eso si.

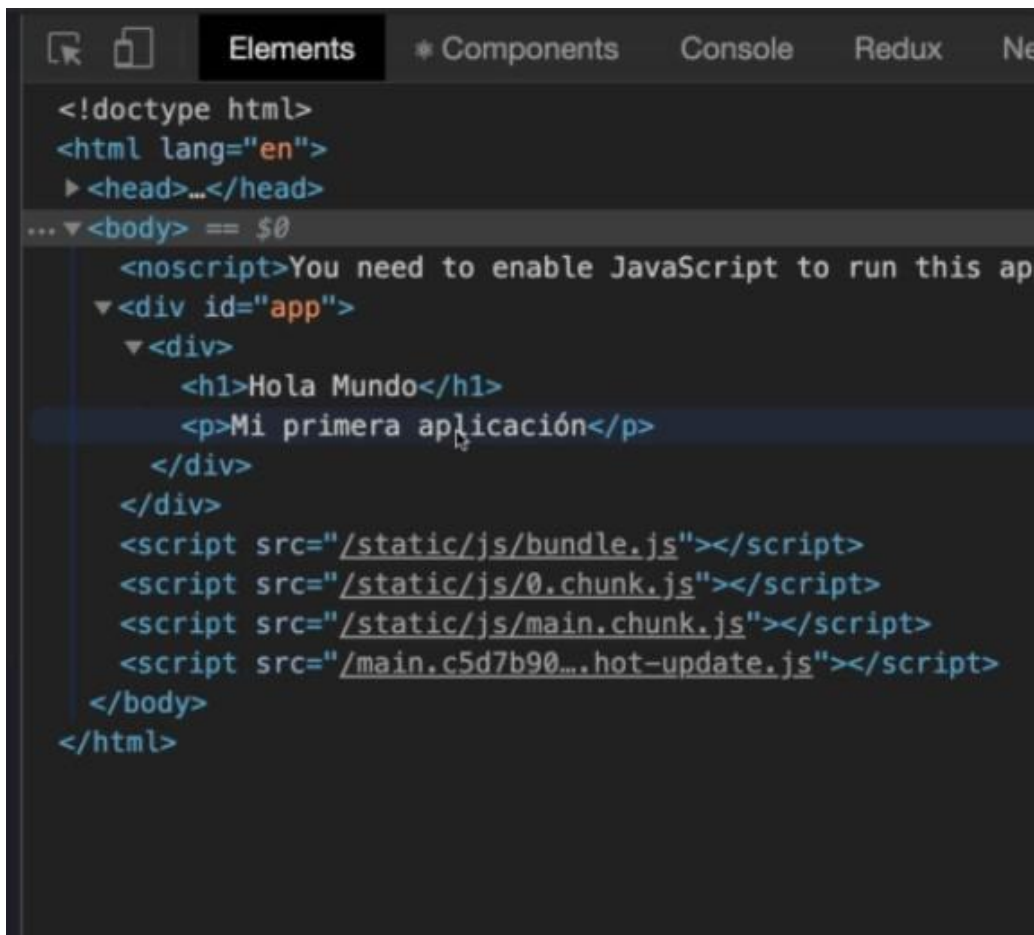
Vamos a utilizar un componente **Fragment** que se denomina *highier old component*(Es un componente que puede albergar otros componentes)

Antes vamos a ver una forma de resolver el problema de mostrar más elementos que podría ser la siguiente:

```
1  import React from 'react';
2
3  const PrimeraApp = () => {
4
5
6      return (
7          <div>
8              <h1>Hola Mundo</h1>
9              <p>Mi primera aplicación</p>
10          </div>
11      );
12
13
14  }
15
16
17  export default PrimeraApp;
```

Esta forma no es la más óptima ya que nos haría meter una etiqueta <div> que realmente no nos es del todo necesaria.

Si nos vamos a la consola y pinchamos en elements veremos lo siguiente:



```
<!doctype html>
<html lang="en">
  <head>...</head>
  <body> == $0
    <noscript>You need to enable JavaScript to run this ap
    <div id="app">
      <div>
        <h1>Hola Mundo</h1>
        <p>Mi primera aplicación</p>
      </div>
    </div>
    <script src="/static/js/bundle.js"></script>
    <script src="/static/js/0.chunk.js"></script>
    <script src="/static/js/main.chunk.js"></script>
    <script src="/main.c5d7b90...hot-update.js"></script>
  </body>
</html>
```

Vamos a tener un `<div>` sin usar.

La forma de solventarlo es utilizar un Fragment.

```
import React, { Fragment } from 'react'
//FC
const PrimeraApp = () => {
  return (
    <Fragment>
      <h1>Hola Mundo</h1>
      <p>Mi primera aplicación</p>
    </Fragment>
  );
}

export default PrimeraApp;
```

Cuando escribimos `<Fragment` si borramos la `t` `<fragmen` + TAB , nos hace la importación de forma automática:

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

Como vemos nos añade el componente { Fragment }

Ahora ya no tenemos el problema del <div> flotante pero quizás esta tampoco sea la manera más óptima ya que nos obliga a hacer una importación adicional.

Vamos a ver otra manera más óptima:

```
import React from 'react';  
// import React, { Fragment } from 'react';  
  
const PrimeraApp = () => {  
  return (  
    <>  
      <h1>Hola Mundo</h1>  
      <p>Mi primera aplicación</p>  
    </>  
  );  
}  
  
export default PrimeraApp;
```

Parece un poco extraña pero así eliminamos el tener un <div> flotante y un <Fragment>.