

# React



**Juan Carlos Pérez Rodríguez**

## Sumario

Introducción.....	4
Instalaciones necesarias.....	4
React.....	5
Empezando de cero nuestra aplicación react.....	8
Reducir tamaño de nuestra App.....	10
Creación de componentes y primeros pasos.....	10
Fragment: devolución en un return de varios elementos.....	11
Incluir variables, expresiones en el return de nuestros componentes jsx.....	13
JSX, TSX.....	14
Pasar propiedades a un componente y verificarlas con PropTypes.....	15
Pasar propiedades a un componente con typescript.....	18
Llamar de un componente a otro con props.....	19
Concepto de State en React. Uso de hooks.....	21
Hooks.....	23
Entendiendo diferencias entre Functional Components (Stateless) y Componentes tradicionales(Statefull). Atributos estáticos.....	27
Pasando información (parámetros ) en un onClick.....	29
Poniendo estilos CSS.....	30
Bucles y condiciones en JSX/TSX.....	31
Bucles e identificador único en Componentes.....	33
Condiciones en JSX/TSX.....	34
hook useEffect().....	36
Hook useRef(). Accediendo al DOM directamente con referencias.....	41
useRef() para atributos sin DOM. Crear y parar un timer setInterval().....	43
Acceder información del DOM mediante eventos, en lugar de referencias.....	45
Pasando información entre componentes hijos y padres.....	49
Multimedia.....	51
Enrutado en React.....	52
Instalación del router.....	53
Haciendo nuestro primer router.....	53
Comunicando con una Api: Axios.....	56
Hook useParams.....	58
axios POST.....	62
Hook useNavigate.....	64
Contexto: Datos compartidos entre componentes.....	65
Seguridad y persistencia en React.....	69
localStorage: persistencia en React.....	70
Enviar token en las cabeceras de las peticiones axios.....	71
Control de acceso a rutas protegidas.....	73

Juan Carlos Pérez Rodríguez

# Introducción

**React** (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. En el proyecto hay más de mil desarrolladores libres. ( definición Wikipedia)

**React Native** es un framework open-source creado por Facebook que se usa para desarrollar aplicaciones Android, IOS, etc Así en lugar de enfocarse al navegador, se hace uso de React en otras plataformas de dispositivos móviles.

React Native llama a las APIs de Android e IOS para hacer el renderizado de la aplicación así hay un “look and feel” de la aplicación React Native como si se hubiera desarrollado en Objective-C ( IOS ) o Java (Android ) Al no usar un webview sino llamar directamente la API nativa es una forma más eficiente que otros sistemas basados en Javascript para desarrollar aplicaciones de dispositivos móviles.

## Instalaciones necesarias

visual studio code, node, postman, android studio, git

extensiones para vscode:

Auto Close Tag;

ES7+ React/Redux/React-Native snippets

JSON to TS; Paste JSON as Code ; Typescript importer

( es posible que sea necesario instalar en el sistema operativo **xclip** )



extensiones de Google Chrome:

React Developer Tools, Redux DevTools

# React

Antes de empezar con React Native precisaremos unos conceptos mínimos de React. Procederemos inicialmente a introducirlos:

Para crear y ejecutar una aplicación react llamada: primeraapp ( nota: se solicita siempre que se usen minúsculas ) podemos usar:

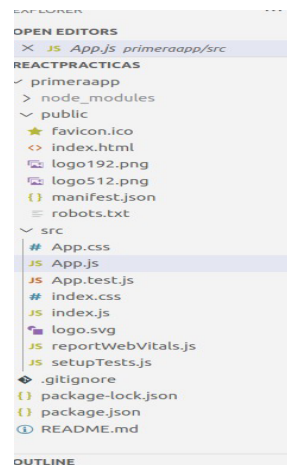
```
npx create-react-app primeraapp  
cd primeraapp ; npm start
```

Y si quisiéramos con **typescript**:

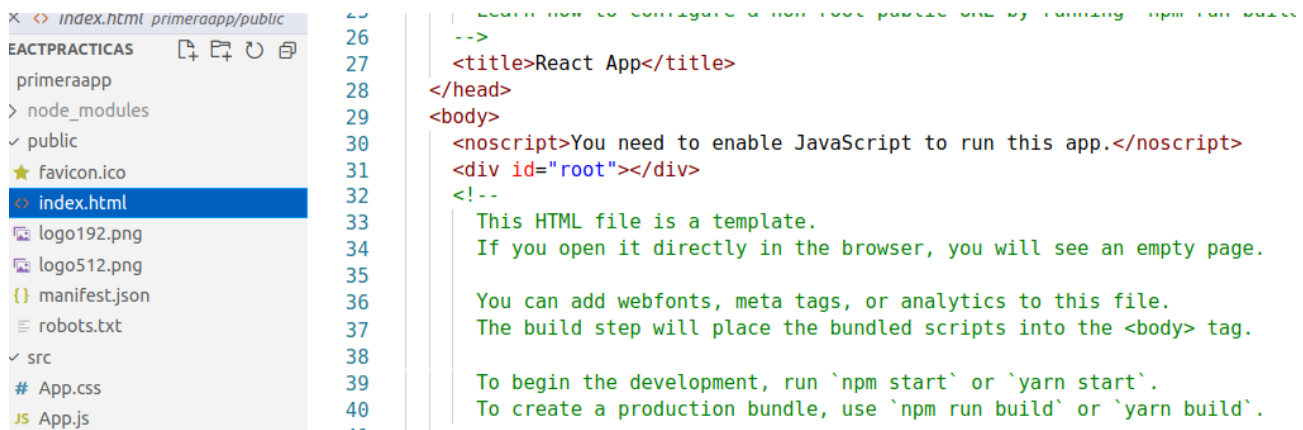
```
npx create-react-app primeraapp --template typescript  
cd primeraapp ; npm start
```

Observamos que nos ha creado una estructura que incluye los módulos de node usados, una carpeta pública y una carpeta: src Es ahí donde vamos a poner nuestra aplicación realmente

La siguiente imagen ilustra la estructura de carpetas



Sabemos que las aplicaciones suelen empezar en algún index. En este caso miremos el **index.html**:



Si leemos lo que se describe en comentarios XML vemos que nos dice que es una aplicación pura de javascript ( no funciona si el navegador no lo tiene habilitado ) y realmente estamos ante un fichero vacío. Únicamente tenemos un espacio contenedor: `<div id="root">` que es el espacio donde se va a renderizar nuestra aplicación

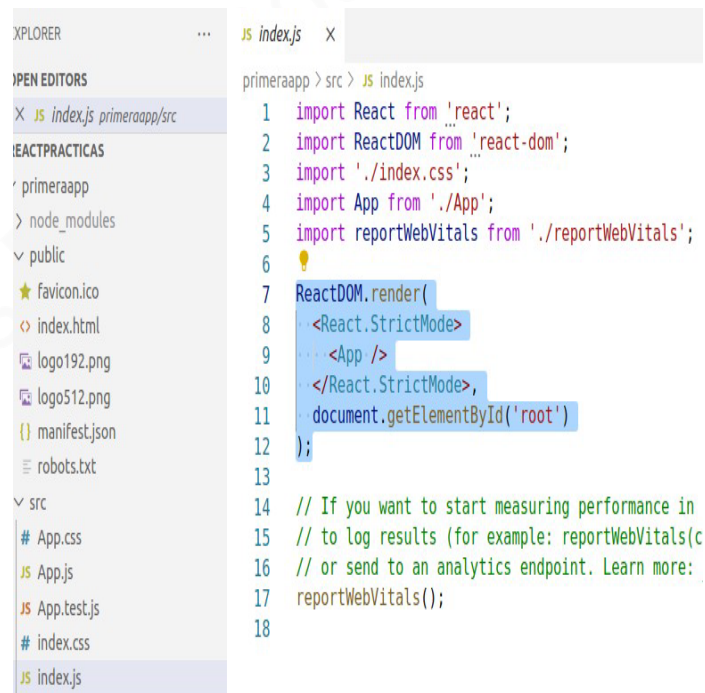
Ahora veamos un fichero más interesante: `index.js`

Vemos que hay un `document.getElementById('root')` que es el `<div id='root'>` que vimos antes en `index.html`

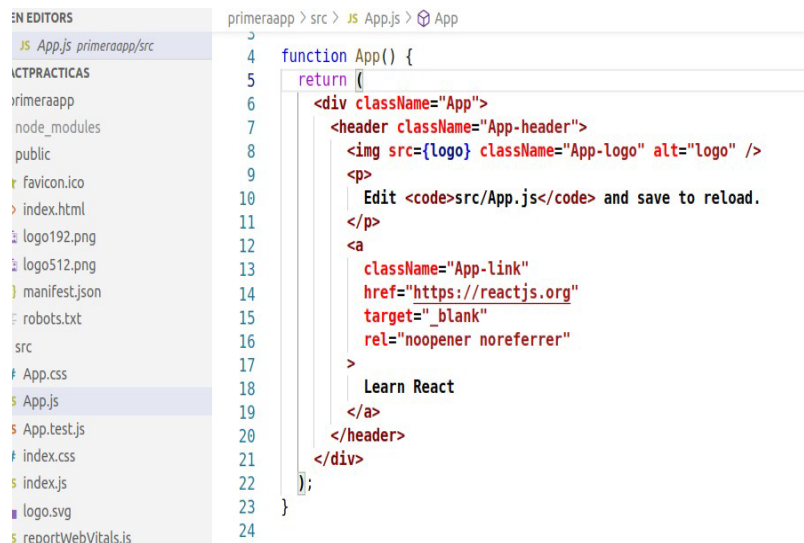
También vemos que hay una etiqueta: `<App />`

Esa etiqueta nos está indicando que ahí hay un componente

que se llama: `App` La idea de trabajar por componentes es tener asociado un objeto con parte gráfica ( código html y estilos css ) y una parte funcional ( atributos y funciones en código javascript ) No difiere de la idea que tendríamos de un objeto componente en otros lenguajes de programación: `textfield` en Java Swing, etc



Finalmente miremos el fichero: **App.js** que realmente es el de nuestra aplicación ( es el componente principal: `<App />` que vimos en index.js )



```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
```

Como una primera aproximación vamos a borrar casi todo lo escrito ahí y haremos un hola mundo en el fichero App.js:

```
function App() {
  return (
    <div className="App">
      <h3> Hola Mundo! </h3>
    </div>
  );
}
export default App;
```

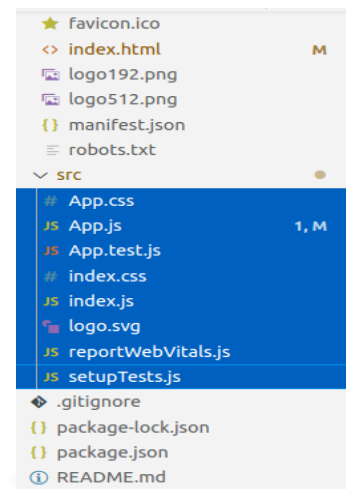
Adicionalmente en index.html vamos a cambiar el title a: `<title> hola mundo </title>`

● **Práctica 1:** Crear el hola mundo descrito y agrega tu nombre completo al `<h3>` (usando `npx` para crear la app y `npm start` para arrancarla como se indica en el tema )

## Empezando de cero nuestra aplicación react

Vamos a borrar el casi todo el contenido de la carpeta: **src** ( recordar que ahí es donde va estar nuestra aplicación ) Básicamente dejamos el package.json, .gitignore y el readme.md

En la imagen se muestra seleccionado lo que vamos a borrar:

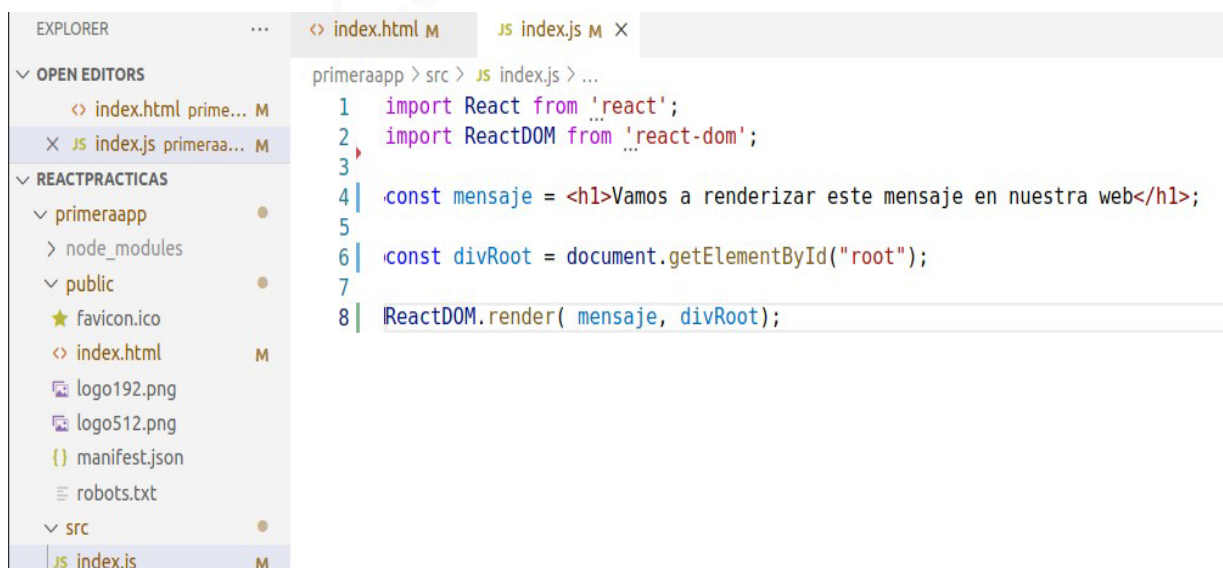


Vamos a crear nuestro propio punto de partida de nuestra app. Para ello hacemos nuestro propio: src/index.js

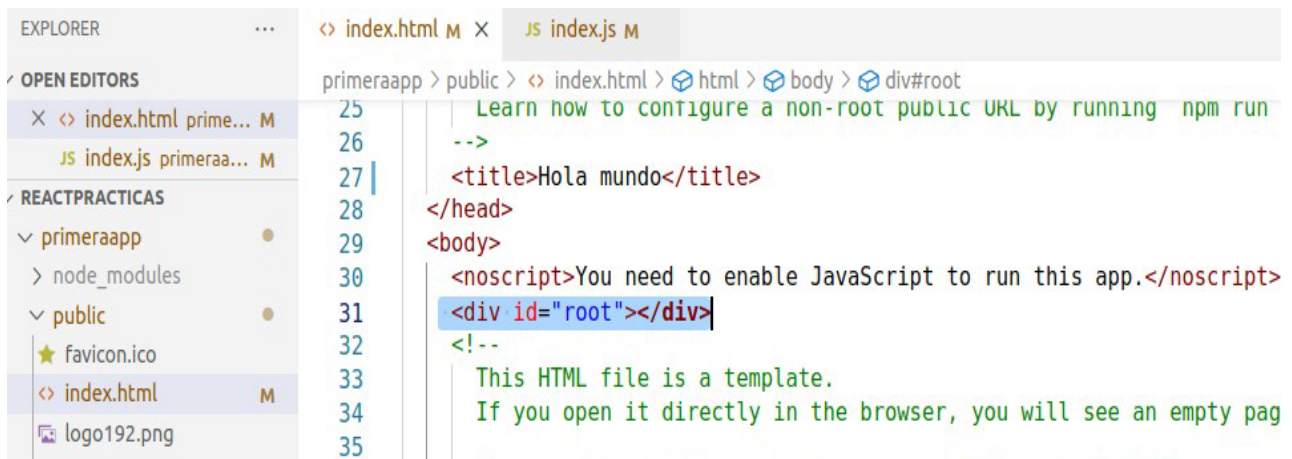
Ponemos el siguiente texto en index.js: ( se muestra en texto copiable y más abajo como imagen )

```
import React from 'react';
import ReactDOM from 'react-dom';// también puede aparecer como: 'react-dom/client'

const mensaje = <h1>Vamos a renderizar este mensaje en nuestra web</h1>;
const divRoot = document.getElementById("root");
ReactDOM.render( mensaje, divRoot);
```







```
25 Learn how to configure a non-root public URL by running npm run
26 -->
27 <title>Hola mundo</title>
28 </head>
29 <body>
30 <noscript>You need to enable JavaScript to run this app.</noscript>
31 <div id="root"></div>
32 <!--
33 This HTML file is a template.
34 If you open it directly in the browser, you will see an empty pag
35
```

Se han puesto las dos capturas para que observemos que cuando ponemos en index.js:

```
const divRoot = document.getElementById("root")
```

Estamos cogiendo el elemento <div> con el id=root que está en index.html

Finalmente vemos que se ejecuta: `ReactDOM.render(mensaje,divRoot);`

Que lo que está diciendo es que nos visualice nuestro “mensaje” en divRoot

Observar que en la constante mensaje no hemos puesto comillas al texto que le asignamos. Queremos que se lo tome como un código html no como un texto

● **Práctica 2:** Realizar lo descrito y tomar captura de pantalla del mensaje en el navegador ( recordar que por defecto la web está en el puerto 3000 )

## Reducir tamaño de nuestra App

Si observamos el tamaño generado en sistema de ficheros de la aplicación observamos que es grande ( más de 200MB ) Eso es principalmente por los módulos de node ( carpeta: node\_modules ) **Para poder exportar nuestra aplicación podemos eliminar esa carpeta.** Para luego volver a dejarla operativa ejecutamos en un terminal que esté dentro de la carpeta de nuestra app:

```
npm install
```

## Creación de componentes y primeros pasos

Vamos a crear un fichero llamado: src/ComponenteApp.js y dentro ponemos:

```
import React from "react";

const ComponenteApp = () => {
  return (
    <h1>Buenos días. Este es nuestro primer componente</h1>
  )
}

export default ComponenteApp;
```

Observar que hemos hecho un import de: React pero no parece que lo estuviéramos usando. Realmente sí, porque es necesario para que nos lo trate como un componente y nos lo renderice

Observar también que en el return NO ponemos lo que devolvemos entre comillas. Queremos que nos lo tome como html ( o mejor dicho jsx ). De hecho, es mejor que tomemos conciencia del uso de los paréntesis en el return: return ( )

En general, será lo aconsejable siempre, ya que le estamos diciendo que queremos devolver un objeto y nos está agrupando las diferentes líneas que pudiéramos escribir en el return como un único objeto a devolver

Ahora para poder usar el componente recién creado vamos a index.js y lo dejamos así:

```
import React from 'react';
import ReactDOM from 'react-dom';
import ComponenteApp from './ComponenteApp';

const divRoot = document.getElementById("root");

ReactDOM.render( <ComponenteApp />, divRoot);
```

Fijarse en que hemos tenido que importar nuestro componente del otro fichero y que ya cuando estamos renderizando ( ReactDOM.render ) ya lo estamos usando como si fuera una nueva etiqueta html que tuviera su propio comportamiento: <ComponenteApp />

## Fragment: devolución en un return de varios elementos

Cuando queremos que nos devuelva varios elementos nuestro componente ( será lo más habitual ) , no basta con poner paréntesis en el return: return ( ) sino que adicionalmente los elementos deben estar agrupados en un único objeto.

**Primera opción:** usando un **<div>**:

```
import React from "react";

const ComponenteApp = () => {
  return (
    <div>
      <h1>Componente con varias líneas</h1>
      <h4>(hace falta una etiqueta contenedora como: div ) </h4>
    </div>
  );
}

export default ComponenteApp;
```

El problema a lo anterior es que se está agregando al html final etiquetas <div> completamente innecesarias, ya que únicamente lo estamos usando para poder hacer un return que contenga el h1 y el h4 a la vez

La solución viene por la etiqueta: **<Fragment>** que es propia de react y así la librería sabe que únicamente queremos agrupar los elementos en un return

El código del return anterior quedaría así:

```
return (  
  <Fragment>  
    <h1>Componente con varias líneas</h1>  
    <h4>(hace falta una etiqueta contenedora )</h4>  
  </Fragment>  
);
```

Habitualmente veremos la **forma “acortada”** de escribir una etiqueta fragment. Que es : **<>**

Ahora el return queda:


```
return (  
  <>  
    <h1>Componente con varias líneas</h1>  
    <h4>(hace falta una etiqueta contenedora )</h4>  
  </>  
);
```

## Incluir variables, expresiones en el return de nuestros componentes jsx

Veamos el siguiente código:

```
const ComponenteApp = () => {  
  const primos = [2, 3, 5, 7];  
  return (  
    <>  
      <h1>Primeros números primos:</h1>  
      <h4>{JSON.stringify(primos)}</h4>  
    </>  
  );  
}
```

Fijémonos primero en las llaves: “{“ y “}” esa va a ser la forma habitual en la que le vamos a decir a react que queremos que nos “interprete” el contenido. Luego simplemente vemos una sentencia Javascript: `JSON.stringify(primos)` que nos está formateando en JSON nuestro array de primos y así poderlo mostrar bien presentado en nuestra página web

-  **Práctica 3:** Reproducir el ejemplo anterior, pero en lugar de mostrar números primos en el `<h1>` dirá: “mis datos:” y en el `h4` le habremos pasado un objeto literal JSON con tu nombre, apellidos y estudios que estás realizando

## JSX, TSX

JSX permite escribir elementos HTML en JavaScript y ubicarlos en el DOM sin necesidad de usar las sentencias habituales para tal cometido en javascript: createElement() o appendChild(). Al manejar html no tenemos que tratarlo como una string. Veamos un ejemplo:

```
export const PruebaApp = () => {  
  const color = 'Verde';  
  const frase = <h4>el color de los árboles es: {color}</h4>;  
  return (  
    <>  
      <h1>Actividad react</h1>  
      {frase}  
    </>  
  )  
}
```

La función está devolviendo un trozo de html, de alguna forma estamos “ampliando” las posibilidades de javascript para integrar html en nuestro código javascript

Vemos que JSX admite variables que se les asigne directamente html ( no está entrecomillado como si fuera un texto ):

```
const frase = <h4>el color de los árboles es: {color}</h4>;
```

Si queremos incluir expresiones javascript dentro de JSX usamos las llaves: {}

```
const color = 'Verde';  
const frase = <h1>el color de los árboles es: {color}</h1>;
```

TSX es lo mismo que JSX pero usando Typescript

JSX y TSX tienen algunas limitaciones respecto a lo habitual en lenguajes de programación. Por ejemplo, los bucles