#### ¿Que es React? Clase 1

[Resumen completo](https://better-domain-e02.notion.site/Curso-Practico-React-95d6f24108bf4dd9b9bb39b210d2d52d)  
React es una librería de JavaScript desarrollada por facebook para crear interfaces de usuario, se enfoca en la vista dentro del modelo vista controlador (MCV).

React propone una arquitectura basada en componentes.

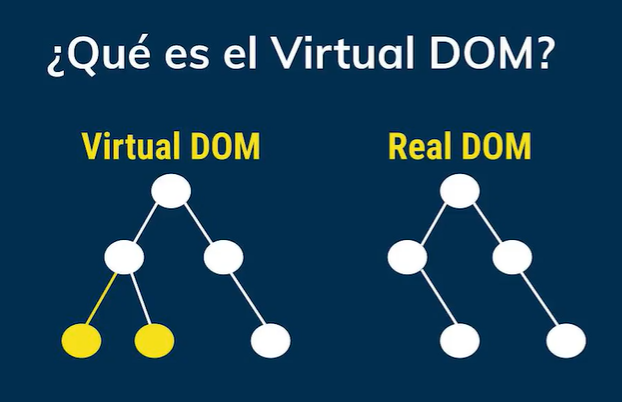
**¿Por qué aprender React?**

Hoy en día, React es una de las librerías más utilizadas y con mayor demanda laboral. Es utilizada por aplicaciones como:Facebook, instagram Netflix, Platzi, Air&Bn, Uber, etc

# Conceptos fundamentales de React Clase 2

**JSX** se refiera a Javascript XML. Nos permite escribir html dentro de Javascript. React no requiere usar JSX, sin embargo la lógica detrás de los componentes, promueven el uso del mismo, porque usamos unidades lógicas centralizadas. En cambio, si tuviéramos diferentes archivos para separar el html y el JS, se perdería la utilidad.

**Virtual DOM** es un concepto precioso de React. El virtual DOM es una copia en memoria del DOM normal del navegador. Pero *¿por qué hacer una copia?* Bueno, cuando el UI cambia de estado, normalmente el DOM es muy lento. Sin embargo, virtual DOM es extremadamente rápido. Lo que ocurre es que primero, todo se computa en el virtual DOM, se actualiza por completo el virtual DOM (a pesar de que se re-renderiza el virtual DOM, no afecta el rendimiento porque primero no se tiene que mostrar gráficamente y realmente es muy rápido) y realiza el proceso de *diff* que compara ambos DOM para después igualar o “reconciliar” ambos UI y cambiar el aspecto de la manera más corta y rápida posible.



**Ciclo de vida** se refiere al proceso de cuando los componentes son creados, actualizados y eliminados. O de otra manera ⇒ nacen, crecen y mueren.

**Estado** son los datos que están dentro del componente. Estos pueden ser actualizados usando diferentes métodos. Los estados son muy importantes en React, pues son encargados de actualizarse cada vez que se cambian, afectando a los demás nodos del virtual DOM.

**Eventos** los componentes, pueden configurarse con eventos como *onclick* para responder antes ciertas interacciones con el usuario, tal como los haríamos en Html

**React Hooks** es otra manera de escribir los componentes con estado, sin usar clases. No se pretenden reemplazar, sin embargo, usar funciones para los componentes pueden facilitar el entendimiento de la aplicación.

#### **¿Qué es el estado en React.js?**

a

El lugar donde almacenamos y actualizamos dinámicamente información de los usuarios y la aplicación.

b

El componente o vista principal que renderiza nuestra aplicación la primera vez que carga.

c

El momento en que la aplicación se compila.

#### **¿Por qué React.js trabaja con el Virtual DOM?**

a

Porque mejora la legibilidad de nuestro código, aunque reduce un poco la velocidad de la aplicación.

b

Porque es el DOM que los navegadores modernos utilizan para ser más rápidos, aunque pierden soporte para aplicaciones viejas.

c

Porque es una copia del DOM real con la que puede hacer cálculos y comparaciones más rápido.

#### **¿Por qué React.js recomienda usar JSX?**

a

Porque es una sintaxis que combina JavaScript con XML, agiliza nuestro desarrollo y mejora la legibilidad de nuestro código.

b

Porque es una sintaxis algo vieja que muchas aplicaciones siguen utilizando, así que vale la pena aprenderla por si nos topamos con ella.

~~c~~

Porque es la única forma de escribir componentes que tengan HTML y JavaScript.

d

Porque es una sintaxis que combina JavaScript con CSS, agiliza nuestro desarrollo y mejora la legibilidad de nuestro código.

La respuesta correcta es:

**Porque es una sintaxis que combina JavaScript con XML, agiliza nuestro desarrollo y mejora la legibilidad de nuestro código.**

#### **Al desarrollar el frontend de una aplicación podemos aplicar el patrón MVC o Modelo, Vista, Controlador. ¿De cuál capa / letra se encarga React.js?**

a

Vista

b

Modelo

c

Controlador

# Análisis de dependencias en proyectos de React (Clase 3)

**LECTURA**

Al desarrollar proyectos de React.js (o de JavaScript con paquetes de NPM en general) es muy importante **analizar las dependencias** con las que vamos a trabajar.

Si creas un proyecto desde cero, lo más seguro es que utilices la última y más moderna versión de tus herramientas favoritas. Pero este no es el caso en la mayoría de proyectos de software “avanzados”.

## ¿Qué version de React utilizar?

¿Has oído de todas las empresas y productos que usan React.js (Platzi, Facebook, Instagram, Netflix, Uber, Airbnb)? Bueno, lamento decirte que ninguna de ellas usa React 18.

¿¡Cómo es posible!? ¿Por qué una empresa de este tamaño no podría actualizarse? ¿Es esto acaso una muestra de incompetencia? ¿Sus equipos de software viven debajo de una piedra? ¿Perdieron interés en nunca parar de aprender?

No. Simplemente actualizar no es una prioridad.

Imagina que trabajas en Platzi. Miles de estudiantes están ansiosos por mejoras en la plataforma. Algunos incluso están desesperados porque tienen un error que les impide avanzar con su aprendizaje.

**¿Qué es más prioritario?**

¿Darle la mejor experiencia posible a tus estudiantes?

¿O migrar de React [inserte versión "vieja" aquí] a React 18, 19, 20 y las que vengan?

Imaginemos otra situación:

Aún no trabajas en Platzi. Estás en entrevista con Diego De Granda. Él está muy satisfecho con tu portafolio y todo lo que sabes de React. Pero te asigna una última prueba para decidir si entras o no al team: resolver todos los bugs que puedas de un proyecto gigantesco que usa React 16.

**¿Qué harías en tu prueba de trabajo?**

¿Decirle a Diego “uy, Dieguillo, qué decepción, yo esperaba más de Platzi, yo no trabajo con versiones cavernícolas de React, ¡adiós! ¡me avisan cuando quieran actualizar!”?

¿O sencillamente trabajar con la React que tenga el proyecto y enfocarte en lo importante (entender la arquitectura, resolver los bugs, etc.)?

## ¿Cómo usar una versión en específico de React?

Para tomar este curso es requisito obligatorio que conozcas cómo trabajar con NPM. Si no lo has hecho antes, te recomiendo tomar el [Curso de Gestión de Paquetes y Dependencias con NPM](https://platzi.com/npm).

Para instalar React y React DOM usamos el comando npm install react react-dom --save. Eso instala automáticamente la última versión de estas tecnologías.

Si quieres usar otra versión, usamos el símbolo @ y la versión con la que queramos trabajar (por ejemplo, en este curso usaremos React 17, así que la instalación puede realizarse con el comando npm install react@17 react-dom@17 --save).

En próximas clases instalaremos React y muchas otras herramientas para configurar nuestro entorno de trabajo profesional. Si te fijas en el archivo package.json, verás que estas son las dependencias con las que trabajaremos en este curso:

"react": "^17.0.2",

"react-dom": "^17.0.2",

"react-router-dom": "^5.2.0",

"webpack": "^5.50.0",

"webpack-cli": "^4.8.0",

"webpack-dev-server": "^4.0.0"

Estás en tu completo derecho de usar estas u otras versiones.

Si usas las mismas que el profesor Oscar, podrás seguir todas sus instrucciones sin conflictos por cambios inesperados en las últimas versiones de React, Webpack o React Router.

Pero también es perfectamente válido instalar la última versión de todo y “cacharrearle” hasta que logres hacerla funcionar. No es el camino más fácil. Pero es parte de la vida. Conseguirlo es un gran logro del que deberías sentir mucho orgullo.

Mi recomendación es que uses las mismas versiones del profesor si es tu primera vez trabajando con estas herramientas. Pero si ya las conoces, tratar de usar versiones diferentes definitivamente será una gran aventura (que vale la pena).

## Recomendación personal

**Recuerda que tú eres el o la máxima responsable de tu aprendizaje.**

Platzi es el camino corto, como hacer trampa en un videojuego para ir más rápido. La increíble comunidad de estudiantes y profes que nos acompaña es maravillosa para hacer tu carrera profesional en cámara rápida.

Pero **tú sigues estando en control de tus decisiones**. Aprópiate de ellas. Toma riesgos y asume sus consecuencias. Lo peor que puede pasar es que debas volver a empezar unas clases atrás. Y eso está muy bien. Vas a avanzar más rápido que la primera vez.

**Siempre nos encontrarás en la sección de aportes y preguntas para ayudarte en todo lo que podamos**. 💚

## ¿Entonces NO vale la pena aprender nuevas versiones de React?

¡No! ¡Esa NO es la lección correcta!

Sí es muy importante conocer los cambios en las nuevas versiones de nuestras tecnologías favoritas. Pero esto no debe impedirte avanzar con tu aprendizaje ni colaborar en proyectos con versiones que aún funcionan aunque no sean la última.

Muchas veces sí aporta muchísimo valor que una empresa o equipo actualice la versión de algunas de sus dependencias:

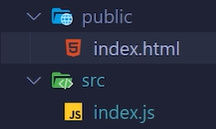
Si ya salió hace un tiempo, es muy probable que los clásicos bugs que aparecen en los lanzamientos ya hayan sido solucionados.

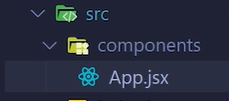
También es posible que esta migración represente una mejora en los tiempos de carga de nuestra aplicación, así que con solo actualizar podemos mejorar la experiencia de los usuarios.

Estas decisiones no suelen tomarse de un día para otro. Se planea con mucho tiempo de anticipación y analizando muy bien qué ventajas a largo plazo traerá esta inversión. A veces puede valer la pena. En otras simplemente no será prioridad.

Entonces, ¿qué versión de React vas a utilizar para el proyecto de React Shop? ¿Cuál será tu estrategia de estudio para completar este curso y sacarle todo el provecho posible? Cuéntame en los comentarios. 😉

# Instalación de React y React DOM (Clase 4)





Codigo de src

// en index.js

**import** React **from**'react';

**import** ReactDOM **from**'react-dom';

ReactDOM.render(<App />, document.getElementById('app'));

// en App.jsx

**import** React **from**'react';

**const** App = () => {

**return** (

<h1>Hola Mundo!</h1>

);

}

Export default App;

<https://github.com/platzi/curso-frontend-developer-practico>

# Configuración de Webpack y Babel (Clase 5)

**Comandos NPM**

Babel:

npm install @babel/core @babel/preset-env @babel/preset-react

Webpack:

npm install webpack webpack-**cli** webpack-dev-server

HTML plugin:

npm install babel-loader **html**-loader **html**-webpack-plugin

* en .babelrc

{ "presets": [

"@babel/preset-env",

"@babel/preset-react" ] }

* En .gitignore

node\_modules

* en webpack.config.js

const path = require('path'); //path del proyecto principal

const HtmlWebpackPlugin = require('html-webpack-plugin'); //traemos el plugin de html

module.exports = {

entry: './src/index.js', // punto de entrada

output: { // lugar al que saldrán todos los archivos

path: path.resolve(\_\_dirname, 'dist'), //en nuestro path, crea la carpeta dist

filename: 'bundle.js' // nombre del archivo js resultante

},

resolve: { // extensión de archivos a tomar en cuenta

extensions: ['.js', '.jsx']

},

module: { // loaders para cada tipo de archivo

rules: [ // reglas para usar

{

test: /\.(js|jsx)$/, // extensiones en las cuales actuará babel

exclude: /node\_modules/, // siempre excluir node modules

use: { // indicamos el loader

loader: 'babel-loader' // babel

}

},

{

test: /\.html$/, // extensiones html

use: [

{

loader: 'html-loader' // loader a usar

}

]

}

]

},

plugins: [ // plugins

new HtmlWebpackPlugin({ // instanciamos el plugin para html

template: './public/index.html', // archivo raíz a transformar

filename: './index.html' // el archivo resultante

})

]

}

# Cambios en tiempo real con Webpack (Clase 6)

En package.json agregar los scripts

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "webpack serve --open",

"build": "webpack --mode production" },

mode: 'development',

En index.js

**import** App **from** './components/App';

# React con CSS y Sass (clase 7)

# npm i mini-css-extract-plugin css-loader style-loader sass sass-loader –D

* en webpack.config.js

**const** MiniCssExtractPlugin = require('mini-css-extract-plugin');

{test: /\.s[ac]ss$/i,

use: [

"style-loader",

"css-loader",

"sass-loader", ], }

**new** MiniCssExtractPlugin({filename: '[name].css' }),

 devServer: {

        allowedHosts: path.join(\_\_dirname, 'dist'), // *contentBase corresponde a webpack 4*

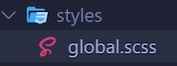
// *ahora en Webpack 5 se usa allowedHosts*

// *créditos al compañero Fabian Rivera Restrepo*

        port: 3005,

        compress: true,

    }



// *creamos la carpeta styles dentro de src*

// *en styles, creamos el archivo global.scss*

$base-color: #ff0000;

$color: rgb(black, 0.88);

body {

    background-color: $base-color;

    color: $color;

}

// *en App.jsx, importamos los estilos de scss*

import '../styles/global.scss';

Ya tienes tu loader de Babel instalado y conectado con Webpack. ¿Dónde defines los presets con los que vas a trabajar?

.babelrc

¿Cuál es el archivo de configuración donde definimos los loaders con los que trabajaremos cada distinto tipo de archivo en nuestro proyecto?

webpack.config.js

¿Cuál es el loader de Webpack necesario para trabajar con código JavaScript interpretado por Babel?

babel-loader

¿Cuál es el comando correcto para trabajar con la versión 17 de React y React DOM.

npm install react@17 react-dom@17 --save

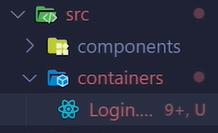
¿Cuál es el comando correcto para instalar la última versión de React y React DOM.

npm install react react-dom --save

## Diferencia entre CSS y SCSS

|  |  |  |
| --- | --- | --- |
| **Parámetros** | **CSS** | **SCSS** |
| Significado y definición | Es un lenguaje de secuencias de comandos, utilizado para desarrollar varias páginas web. | Es una versión más evolucionada y avanzada del lenguaje CSS. Es un tipo de lenguaje de preprocesador, y necesitamos compilarlo o interrumpirlo en el lenguaje CSS. |
| Líneas de códigos | CSS hace un uso extensivo de líneas de códigos. | SCSS utiliza comparativamente menos líneas en sus códigos. |
| Funciones | CSS consta de varias características comunes y, por lo tanto, puede realizar algunas funciones básicas. | SCSS consta de características más avanzadas y, por lo tanto, puede realizar funciones más avanzadas. |
| Reglas para anidar | Un lenguaje CSS regular no asigna varias reglas anidadas. | El lenguaje SCSS promueve reglas que están correctamente anidadas. |
| Diseño | CSS es un lenguaje de estilo que permite a los usuarios crear, diseñar y dar estilo a varias páginas web. | SCSS es un tipo de archivo especial en un programa SASS que uno necesita escribir en el lenguaje Ruby. |
| Idiomas utilizados | Generalmente usamos CSS en los lenguajes JavaScript y HTML. | Generalmente usamos el SCSS en el lenguaje Ruby. |

# Transformando HTML y CSS en componentes de React (clase 8)



Instalar la extexión: React-Native/React/Redux snippet

Para usar **imp** y **slr**

# Estilos en los componentes de React (clase 9)

Instalar la extexión ES7+ React/Redux/React-Native snippets

# React Router DOM (clase 10)

#### ¿Qué es router en React?

Debido a que React es de tipo SPA(single page application), no recarga la página cuando cambiamos de url. Sin embargo, router nos ayuda a crear otra página para poder navegar en nuestra aplicación. Imagina twitter web, cuando das click en un tweet, se abre otra sección donde puedes ver el tweet. Sería un problema que al momento de darle click, no cambie la url, por lo que ese tweet no tiene dirección propia, no se guardaría en tu historial y sería un problema el SEO. Para ello, usamos router, que se encargará de administrar esta situación, donde en el momento que abras el tweet, cambie la URL, pero todavía mantenga ese dinamismo y rapidez de una SPA.

#### ¿Entonces qué es ReactRouterDOM?

#Para instalar

npm install react-router-dom

//import en App.jsx

**import** { BrowserRouter, Switch, Route } **from** 'react-router-dom';

// usaremos esas 3 herramientas

**ReactRouterDOM** te permite implementar enrutado dinámico en la aplicación. Nos facilita pues podemos enrutar nuestra app basada en componentes de la app (como login o recoverypassword).

**const** App = () => {

**return** (

<BrowserRouter>

<Switch>

<Layout>

<Route exact path="/" component={Home} />

<Route exact path="/login" component={Login} />

<Route exact path="recovery-password" component={Recoverypassword} />

<Route component={NotFound} />

</Layout>

</Switch>

</BrowserRouter>

);

}

#### ¿Qué estamos haciendo?

**BrowserRoute** sirve para implementar router en el navegador

**Switch** regresa la primera ruta que coincida. En pocas palabras, si estamos en [*www.platzi.com/contacto*](https://platzi.com/clases/2484-react-practico/42074-react-router-dom/platzi.com/contacto) , regresará el componente que coincida a este (es decir, el componente que contenga la lógica de contacto). En esta caso, estamos poniendo varios routes dentro de switch, ¿para qué? para que solamente traiga esa misma ruta, y no tenga que buscar más. Como si fuese un condicional switch de javascript efectivamente. Y por ello tenemos un route sin path, que será el valor por defecto.

**Layout** solamente renderizará el route que coincida efectivamente con la URL especificada.

# Navegación entre rutas (clase 11)

Muy importante, a partir de las mas nuevas versiones de ‘react-router-dom’ el Switch fue cambiado por ‘Routes’ entonces el import quedaria asi"

**import** { BrowserRouter, Routes, Route } **from** 'react-router-dom';

Tambien en vez de "component en el atributo de Route, ahora es “element” y se le pasa el elemento, la estructura asi:

<BrowserRouter>

<Layout>

<Routes>

<Route exact path="/" element={<Home />} />

<Route exact path="/login" element={<Login />} /> <Route exact path="/recovery-password" element={<RecoveryPassword />} />

<Route path="\*" element={<NotFound />} />

</Routes>

</Layout>

</BrowserRouter>

# Header en todas las rutas (clase 12)

Instalar la extensión ES7 React/Redux/GraphQL/React-Native snippets para poder formatear el código