educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 1

Creando un proyecto

Preparar el entorno de desarrollo

En este manual, vamos a preparar **el entorno de desarrollo para ReactJS.**

Cuando nos preparamos, siempre lo hacemos **pensando en un momento concreto**. Tomemos algunos ejemplos cotidianos: cuando nos vestimos y arreglamos, lo hacemos para salir a un encuentro determinado; cuando preparamos nuestro portfolio, esto es para estar preparados para el momento en que debamos mostrarlo a un posible cliente o empleador.

De la misma forma, cuando preparamos un entorno de desarrollo, lo hacemos **orientados a un momento concreto: cuando se construye la aplicación final**. Es decir, lo preparamos para el ***build time***. El resto de las demás características son, ciertamente, secundarias.

Es por eso que, para entender cómo preparar nuestro entorno de React, tenemos que revisar **cómo se construye el resultado final**.

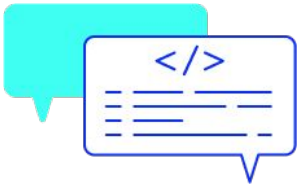
Transpilación

En el primer manual, introdujimos *JSX* y comentamos que el navegador **no entiende de forma nativa *JSX*** sino que sus instrucciones son traducidas en llamadas a **`React.createElement`**.

El código React (JSX + JS) es pasado por un proceso denominado **transpilación** que consiste en **producir un código en lenguaje a partir de otro**.

En React, vamos a usar los paquetes ***Webpack*** y ***Babel*** para ejecutar este proceso.

Dichos paquetes son gestionados automáticamente por la herramienta **create-react-app**, desarrollada por *Facebook* para iniciar un nuevo proyecto React.



Compilación	**Interpretación**	**Transpilación**
Genera **un código ejecutable optimizado**. El código fuente **no es visible desde el ejecutable**.	No genera **ningún archivo extra.** El código fuente **es el mismo ejecutable**, por lo que es totalmente visible.	Genera **código fuente en otro lenguaje**. El código fuente **no es visible desde el ejecutable**, ya que éste último está escrito en otro lenguaje de programación.

Crear un proyecto con React

Vamos a empezar por lo más práctico y después iremos revisando las herramientas involucradas en el “detrás de escena”. En definitiva, vamos a estar creando nuestros proyectos con una herramienta denominada **create-react-app**.

Create-react-app es un paquete de *Node* **autoejecutable** que sirve para **generar de forma automática toda la estructura de archivos y carpetas**, con sus dependencias necesarias instaladas, **para empezar a desarrollar cualquier aplicación React**, sin importar su alcance.

Create-react-app

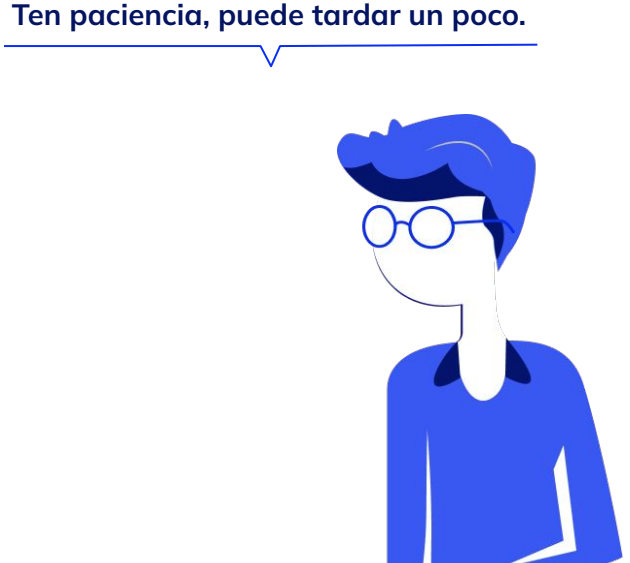
1. Para usar esta herramienta, primero **debemos instalarla** ejecutando el siguiente comando en la consola:

   ```
   npm i -g create-react-app
   ```

2. Una vez instalada, podrás crear **un nuevo proyecto** navegando hacia la carpeta donde quieres que se encuentre y ejecutando allí el siguiente comando:

   ```
   npx create-react-app NOMBRE
   ```

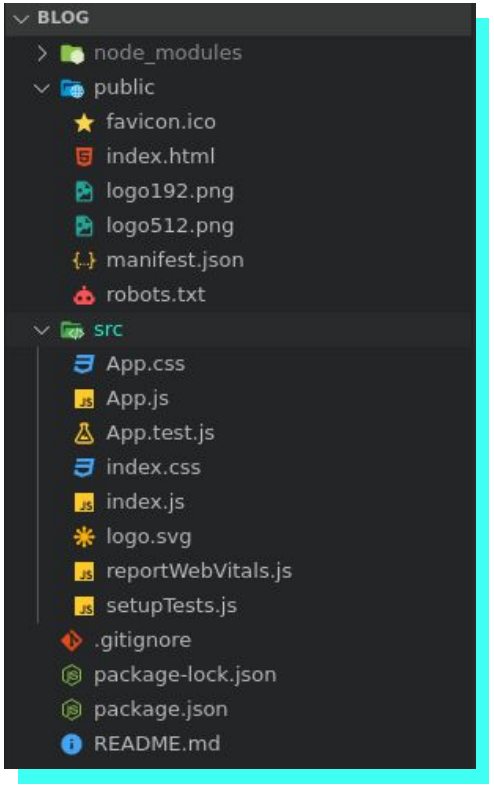
3. Reemplaza **NOMBRE** por el **nombre de tu proyecto sin espacios.**



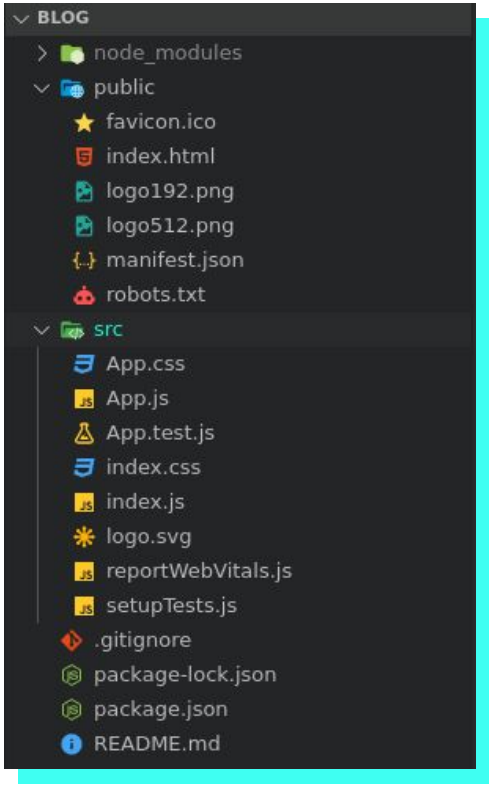
Una vez concluida la instalación, tendremos una estructura de archivos y carpetas similar a la de la imagen que vemos a la derecha.

Hay cuatro partes principales:

- **Las dependencias de *Node*** se encuentran en la carpeta **/node_modules**. Esta carpeta es manejada directamente por ***NPM*** y de nuestra parte **no debemos tocarla directamente**, sino únicamente a través de este programa.
- La carpeta **/public** contiene todos los archivos y carpetas que **son copiados directamente** a la carpeta del resultado final, sin procesar. Aquí hay **imágenes, archivos estáticos y hojas de estilo externas.**



- La carpeta **/src** contiene todos los archivos y carpetas que **no son copiados directamente, sino que son transpilados**. Acá irá **el código React y todo código que será procesado**.
- Luego se encuentran **los archivos de configuración global**, que sirven para **configurar el entorno de desarrollo**.



Webpack

Webpack es un **empaquetador de módulos.** Significa que, con W*ebpack*, puedes convertir **varios archivos separados en un solo archivo que haga exactamente lo mismo.** Por ejemplo:

Al trabajar con *Webpack*, podemos dividir el código en **varios archivos**. Solo podrás incluir (`import`) **código previamente exportado** (`export`) **por otro archivo**.

En este ejemplo, tienes **un archivo principal** (`index`) que requiere, es decir que importa, **dos variables de otro archivo.** Luego de eso, las usa para mostrar la suma por consola.

```
// variables.js

export const numero1 = 20;
export const numero2 = 30;


// index.js

import { numero1, numero2 } from './variables.js';

console.log("La suma es "+(numero1 + numero2));
```

Al llegar el momento de construir el programa, de generar el resultado final que será entregado, *Webpack* empieza a analizar **desde el punto de entrada** (normalmente es el archivo `index`).
De ese archivo analiza los `imports` y de esos archivos requeridos sus `imports` y así arma el árbol de dependencias.

Una vez creado el árbol, genera un código que hace **exactamente** lo mismo, pero **ofuscado** y **optimizado** (lo empaqueta).

En este ejemplo, el resultado final sería el siguiente:

```
// main.js

console.log("La suma es "+50);
```

Existen muchas formas de trabajar con Webpack. En principio, está el paquete ***NPM webpack***, que es la base del resto de paquetes. Sobre el paquete principal se montan los llamados ***plugins***.

Un *plugin* es **un agregado a este proceso de compilación**.

Puedes hacer cosas como:

- Generar **un *HTML* para contener el *JS* resultante.**
- Generar **un *ServiceWorker* que almacene en caché los archivos generados por *Webpack*.**
- **Comprimir imágenes.**
- **Minificar el código.**

> Existen muchas cosas que puedes hacer con Webpack. En su web tienes una lista completa de plugins y opciones de configuración: webpack.

Babel

Si bien *Webpack* se encarga de compilar todos los archivos necesarios ¿Qué sucede con *JSX*? ¿Bastará con que se unan varios *JSX* en uno solo para que eso lo entienda el navegador?
La respuesta es que no. Para eso **necesitamos otro programa que trabaje junto con *Webpack* para convertir el código *JSX* en llamadas a `React.createElement`**. Esa herramienta es ***Babel*** y justamente esa es su función.

Vamos a usar *Babel* **en el momento en que desarrollemos en un código distinto al procesable por el navegador.**

En el caso de React, ese código es *JSX*. Así que, **siempre que usemos *JSX*, necesitaremos *Babel*.**

Create-react-app se configura **automática-mente** para usar *JSX*, por lo que no necesitas instalar nada. Sin embargo, puedes investigar más esta herramienta en su página oficial.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!