

# Principales herramientas que instalamos con Create React App

(además de React y ReactDOM)

**DigitalHouse** >  
Coding School



**Certified Tech  
Developer**

The Ultimate Degree

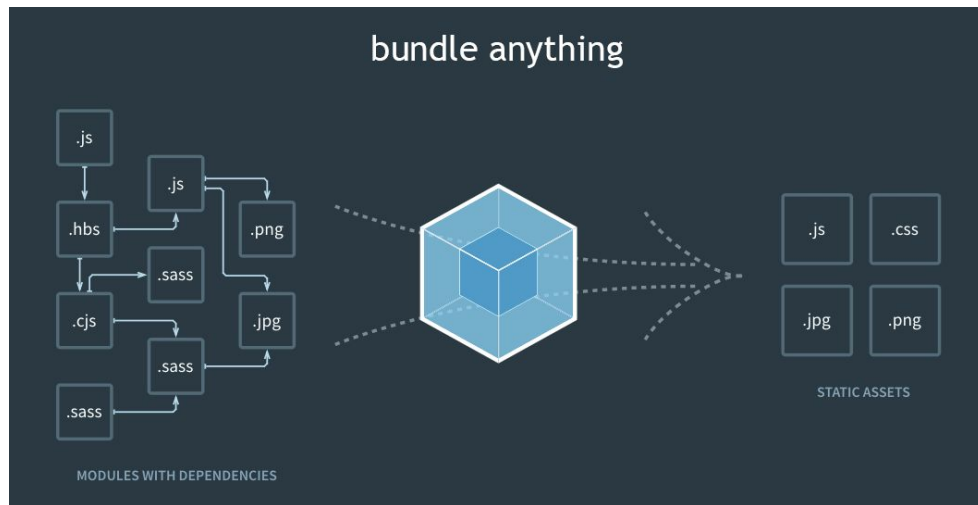
# Índice

1. [Webpack](#)
2. [Babel](#)
3. [ESLint](#)
4. [Jest](#)
5. [@testing-library](#)

# 1 | Webpack

# Un empaquetador de archivos

Webpack es un empaquetador de aplicaciones que convierte archivos y módulos en un estático distribuible. En términos más simples, es un bundle, un conglomerado de varios recursos que una página web necesita para funcionar.

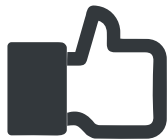




El componente obvio de un bundle es Js, pero en algunos casos también incluye CSS e imágenes, codificadas para su uso en data URLs.



El bundle frecuentemente es solo un archivo.



Beneficios de un bundle:

- Solo requiere una solicitud a la red.
- Puede superponer otras optimizaciones como la minificación y compresión de código.

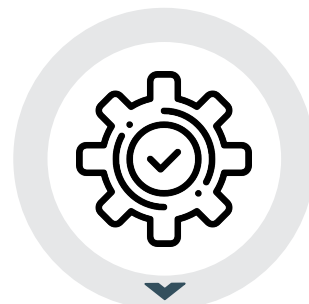
Cuando se le dice a Webpack que empaquete una aplicación, este mira todos los archivos que encuentra, examina y determina qué archivos dependen de otros. Crea lo que se llama un gráfico de **dependencias bundles** de forma inteligente para tener la cantidad mínima de código requerida para que un sitio web o aplicación web funcione.



**Empaqueta  
código**



**Transpila  
TypeScript**



**Sabe cómo  
trabajar con  
React y sus  
archivos .tsx**

Lo mejor de Create React App es que nos evita tener que configurar una aplicación por nosotros mismos.

Veamos un ejemplo de todo lo que hace Create React App con Webpack sin que tengamos que preocuparnos.

```
use strict;

const autoprefixer = require('autoprefixer');
const path = require('path');

const webpack = require('webpack');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const CaseSensitivePathsPlugin = require('case-sensitive-paths-webpack-plugin');
const InterpolateHtmlPlugin = require('react-dev-utils/interpolateHtmlPlugin');
const WatchMissingNodeModulesPlugin = require('react-dev-utils/WatchMissingNodeModulesPlugin');
const defineFormatter = require('react-dev-utils/formatFormatter');
const ModuleScopePlugin = require('react-dev-utils/ModuleScopePlugin');
const getClientEnvironment = require('./env');
const paths = require('./paths');

// Webpack uses 'publicPath' to determine where the app is being served from.
// It defaults to the root of the page where the JS bundle is loaded.
// In development, we always serve from the root. This makes config easier.
const publicPath = '/';

// 'publicUrl' is just like 'publicPath', but we will provide it to our app
// as %PUBLIC_URL% in "index.html" and 'process.env.PUBLIC_URL' in
// javascript.
// Omit trailing slash as %PUBLIC_PATH%/xyz looks better than
// %PUBLIC_PATH%xyz.
const publicUrl = '';

// Get environment variables to inject into our app.
const env = getClientEnvironment(publicUrl);

// This is the development configuration.
// It is focused on developer experience and fast rebuilds.
// The production configuration is different and lives in a separate file.
module.exports = {
  // You may want 'eval' instead if you prefer to see the compiled output
  // in DevTools.
  // See the discussion in https://github.com/facebookincubator/create-react-app/issues/543
  mode: 'development',
  // 'web' extension prefixes have been added for better support
  // for React Native Web.
  extensions: ['.web.js', '.mjs', '.js', '.json', '.web.jsx', '.jsx'],
  alias: {
    // Support React Native Web
    // https://www.smashingmagazine.com/2016/08/a-glimpse-to-future-with-react-native-for-web/
    'react-native': 'react-native-web',
  },
  plugins: [
    // prevents users from importing files from outside of src/ (or
    // node_modules)
    // This often causes confusion because we only process files within
    // src/ with babel.
    // To fix this, we prevent you from importing files out of src -- if you'd
    // like to,
    // please link the files in your node_modules/ and let module-
    // loader look in there.
    // Make sure your source files are compiled, as they will not be
    // processed in any way.
    new ModuleScopePlugin(paths.appSrc, [paths.appPackageJson]),
  ],
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        loader: 'babel-loader',
        options: {
          // TODO: Disable require.ensure as it's not a standard language
          // feature.
          // We are waiting for https://github.com/facebookincubator/create-react-app/issues/2176.
          // { parser: { requireEnsure: false } },
        },
      },
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader',
        ],
      },
    ],
  },
  resolve: {
    // This helps to find the 'react' and 'react-dom' packages
    // instead of finding the 'babel-core' package (we'll add this
    // later).
    // If you want to use other packages, like 'jquery' or 'moment',
    // add them to this list.
    modules: [
      paths.appSrc,
      paths.appNodeModules,
      paths.resolveModuleNames,
    ],
  },
  devServer: {
    contentBase: paths.appPublic,
    compress: true,
    port: 3000,
  },
  proxy: {
    // Proxy for all requests to this configuration.
    // See https://github.com/facebookincubator/create-react-app/issues/1601
    // for more details.
    // This should only be used if you are using webpack-dev-server.
    '/api': {
      target: 'http://localhost:5000',
      changeOrigin: true,
      pathRewrite: {
        '^/api': '',
      },
    },
  },
};
```

Por defecto, este código está oculto. Pero, para poder acceder a él, podemos ejecutar `npm run eject`.

## 2 | Babel



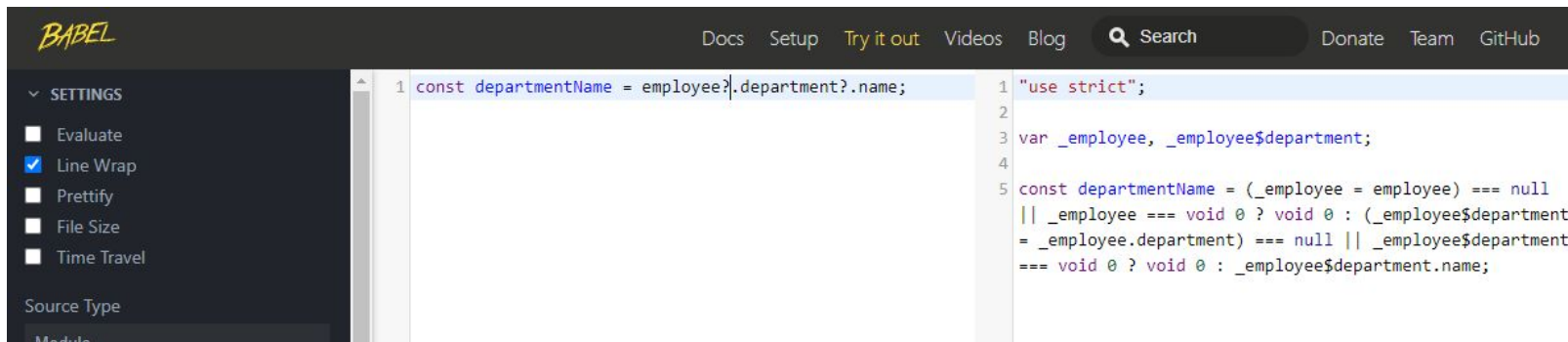
# Un traductor de archivos

Babel es un transpilador/"traductor" que al mismo tiempo:

- Transforma
- Compila



Cualquier código Javascript escrito en versiones superiores a ECMAScript 2015+ a versiones de Javascript que los distintos navegadores del mercado puedan leer.



# Un traductor de archivos

Cuando aparece una nueva versión de ECMAScript, los navegadores y el motor Js de Node necesitan tiempo para absorberla. Babel permite esto: tomará el código y generará un *polyfill* (refactorizar una característica nueva de un lenguaje utilizando características más antiguas de este).

# ¿Qué es ECMAScript?

Es es un estándar publicado por Ecma International. Contiene la especificación para un lenguaje de scripting de propósito general. Es decir, sirve como guía o referencia para estandarizar los avances del código de Javascript.



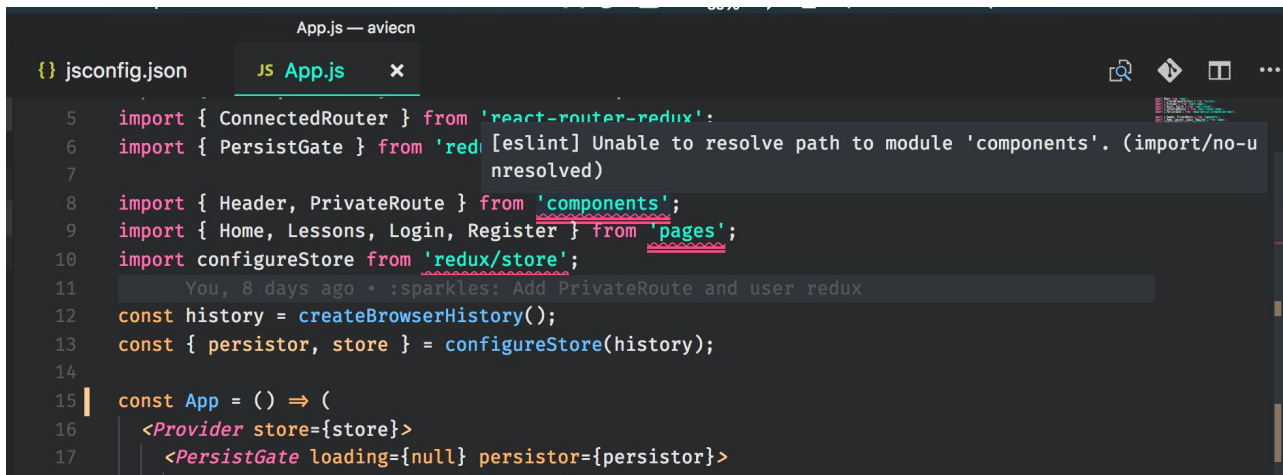
Hacé [clic acá](#) para ver un ejemplo creado en Babel.

A la izquierda podemos ver el uso de optional chaining y a la derecha cómo Babel transforma el código a una versión más antigua de Javascript.

# 3 | ESLint

# ESLint

ESLint es una herramienta de linting. Realiza análisis de código estático para identificar patrones problemáticos encontrados en el código JavaScript. Fue creado por Nicholas C. Zakas en 2013.



The screenshot shows a code editor with a file named 'App.js' open. The code is written in JavaScript and includes imports from 'react-router-redux', 'redux', 'components', and 'pages'. There is an ESLint error message displayed: '[eslint] Unable to resolve path to module 'components'. (import/no-unresolved)'. The error is pointing to the import statement for 'components'. The code also includes a comment: 'You, 8 days ago • :sparkles: Add PrivateRoute and user redux'. The code is as follows:

```
5 import { ConnectedRouter } from 'react-router-redux';
6 import { PersistGate } from 'redux';
7
8 import { Header, PrivateRoute } from 'components';
9 import { Home, Lessons, Login, Register } from 'pages';
10 import configureStore from 'redux/store';
11
12 const history = createBrowserHistory();
13 const { persistor, store } = configureStore(history);
14
15 const App = () => (
16   <Provider store={store}>
17     <PersistGate loading={null} persistor={persistor}>
```

# 4 | Jest

“

**Jest es un marco de prueba de JavaScript mantenido por Facebook, Inc. diseñado y construido por Christoph Nakazawa con un enfoque en la simplicidad y el soporte para grandes aplicaciones web.**

”

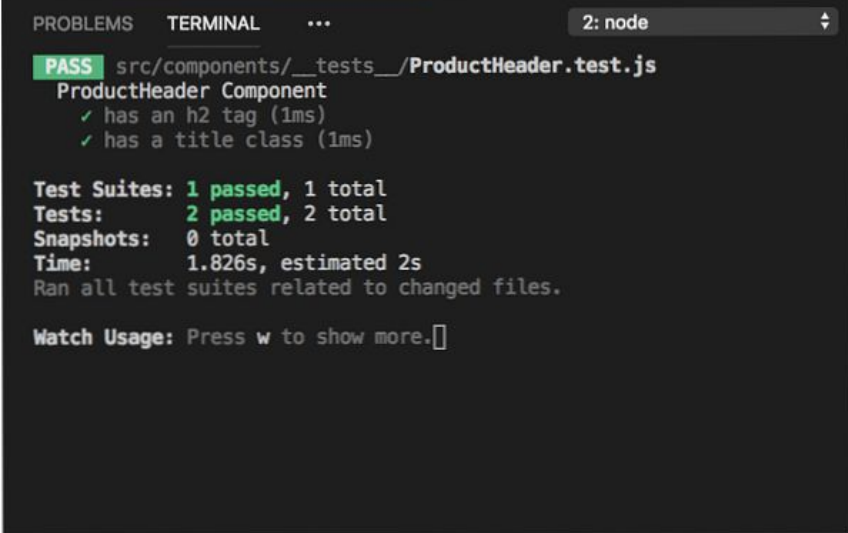


# 5 | @testing-library



# @testing-library

React Testing Library es una solución muy liviana para probar componentes de React. Proporciona funciones de utilidad ligeras con el fin de fomentar mejores prácticas de testeo. Su principio rector principal es: "Cuanto más se asemejen sus pruebas a la forma en que se utiliza su software, más confianza le pueden dar."



```
PROBLEMS  TERMINAL  ...  2: node
PASS src/components/__tests__/ProductHeader.test.js
  ProductHeader Component
    ✓ has an h2 tag (1ms)
    ✓ has a title class (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.826s, estimated 2s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

DigitalHouse>  
Coding School