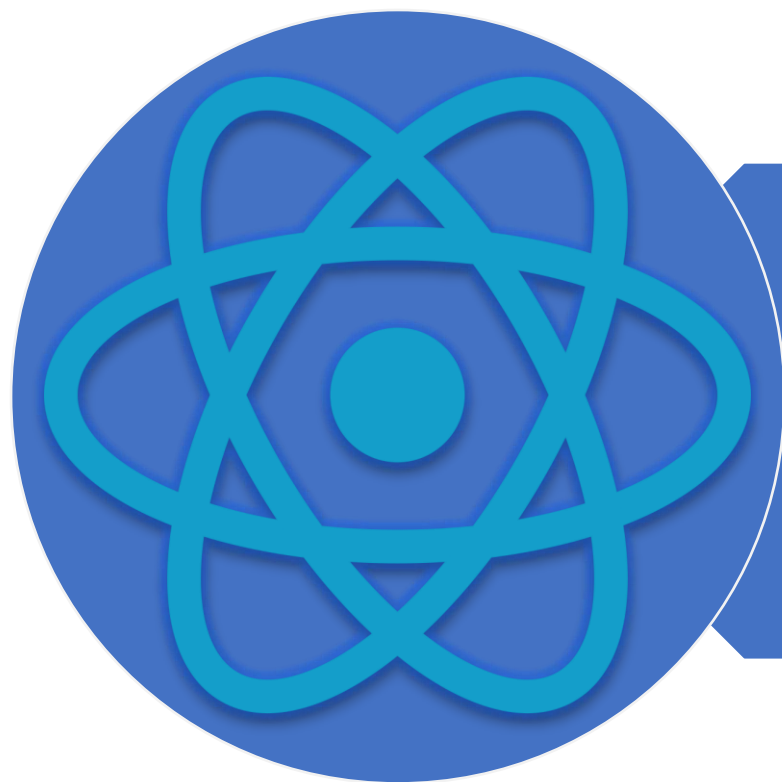


# Desarrollo Web en Entorno Cliente

## Tema 20





# Introducción a React



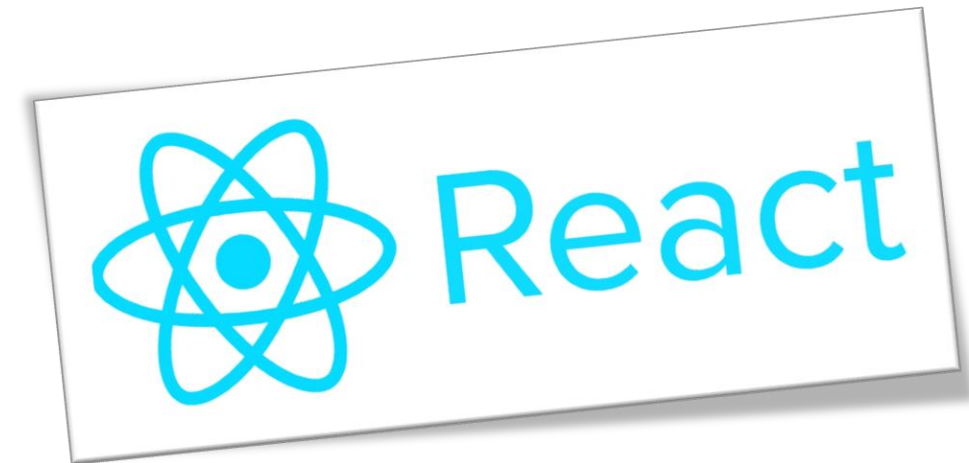
# [React]

React es una librería muy popular de JavaScript para el desarrollo de aplicaciones móviles y web.

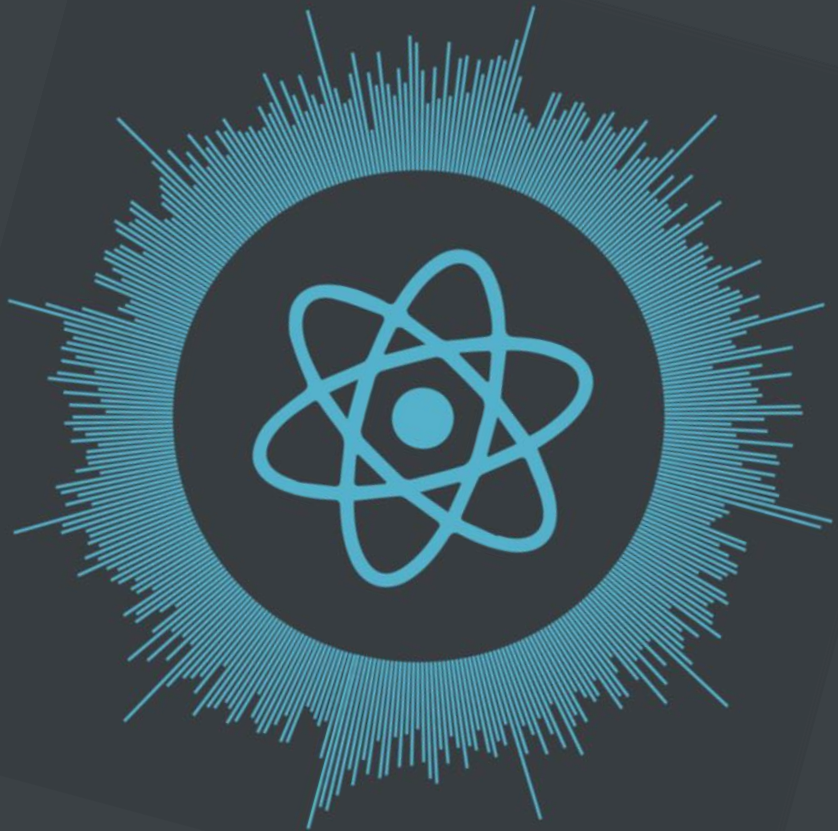
Fue creado por la empresa Meta para proporcionar una mejor estructura a su código base y permitirle escalar mucho mejor, funcionó tan bien en Facebook que la hicieron de código abierto.

Actualmente, React es la principal tecnología para construir la interfaz de una aplicación; permite construir componentes pequeños, aislados y altamente reutilizables que pueden combinarse para crear interfaces complejas.

Es utilizado por muchas compañías, como Netflix, Uber y Airbnb.



# [React]



React se utiliza para desarrollar componentes de interfaz de usuario.

Los componentes son piezas de la interfaz de usuario que se pueden combinar para crear una interfaz.

Además, los componentes pueden ser reutilizables para que puedan usarse en diferentes páginas o pantallas o incluso en otras aplicaciones.

Esto permite a React incorporarse a una aplicación existente, incluso si utiliza un “framework diferente”, ya que no necesita hacerse cargo de toda la aplicación, solo se ocupa de la parte de la interfaz de la aplicación.

Como con cualquier librería, se puede trabajar con React accediendo a la misma desde html, de la misma manera que se trabaja con MomentJS, pero no es la forma “correcta”.

# [Componente de React]


Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables y, así, poder trabajar en cada una de esas piezas de forma aislada.

Los componentes son como las funciones de JS. Aceptan entradas arbitrarias (llamadas “*props*” que proviene de propiedades) y retornan elementos de React que describen lo que debe aparecer en la pantalla.

Tienen su propia lógica y apariencia, y pueden ser tan pequeños como un botón o tan grandes como toda una página.

El nombre de los componentes, por convención, se ponen en Pascal Case (o *Upper CamelCase*).

Un componente puede tener estado o no.

**COMPONENTES EN REACT** 

Los componentes son piezas de código reutilizables.

P.ej: El componente `<Cupcake />` nos permite crear cupcakes de diferentes sabores y colores.

`<Cupcake color="rosa" sabor="vainilla" />`

`<Cupcake color="azul" sabor="chocolate" />`

Color y sabor son propiedades. "PROPS"

Los componentes pueden tener **estado**. Valores que pueden cambiar. P.ej: `{vendido: true}`

**Se pueden escribir de 2 formas:**

**Función**

```
function Cupcake({color, sabor}) {  
  return (  
    <p  
      className={color}>{sabor}</p>  
  )  
}
```

**Clase**

```
class Cupcake extends Component {  
  render() {  
    return (  
      <p  
        className={this.props.color}>  
        { this.props.sabor }  
      </p>  
    )  
  }  
}
```

Cada vez **se usa menos**.



# COMPONENTES EN REACT



Los componentes son piezas de código reutilizables.

P.ej: El componente `<Cupcake/>` nos permite crear cupcakes de diferentes sabores y colores.



```
<Cupcake color="rosa" sabor="vainilla" />
```

```
<Cupcake color="azul" sabor="chocolate" />
```



Color y sabor son propiedades.  
"PROPS"

Los componentes pueden tener **estado**. Valores que pueden cambiar. P.ej: `{vendido: true}`

Se pueden escribir de 2 formas:

Función

```
function Cupcake({color, sabor}) {  
  return (  
    <p  
      className={color}>{sabor}</p>  
  )  
}
```

Clase

```
class Cupcake extends Component {  
  render() {  
    return (  
      <p  
        className={this.props.color}>  
          { this.props.sabor }  
      </p>  
    )  
  }  
}
```

Cada vez se usa menos.

## Componente de React

Mientras que las *props* son los datos que se pueden pasar a un componente, el estado de un componente, también conocido como *state*, se corresponde con los datos internos que maneja el componente.

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

A medida que estos datos son modificados, ya sea por una interacción del usuario o por una recepción de datos de la API, etc. el estado se va modificando.

Cada cambio de ese estado provoca que el componente se renderice de nuevo con una nueva representación en pantalla.

# JSX

JSX es la sintaxis usan los componentes de React y se utiliza para definir lo que debe mostrar el componente.

JSX es *JavaScript Syntax Extension* ( $JSX = JS + XML$ ), una extensión de la sintaxis de JavaScript desarrollada por Facebook para React. Es un lenguaje de plantillas que muestra el HTML pero tiene la funcionalidad de JS.

No se ejecuta directamente en el navegador; primero debe transpilarse a JS utilizando, por ejemplo, Babel.

En el siguiente ejemplo se puede apreciar una mezcla sintaxis entre JS y HTML (XML):

```
const h1Tag = <h1> Hola mundo, soy {nombre}</h1>;
```

Los ficheros que contienen código JSX tienen normalmente extensión `.jsx`.



# JSX



La ventaja de JSX es que simplifica la sintaxis de JS para generar código HTML, por ejemplo:

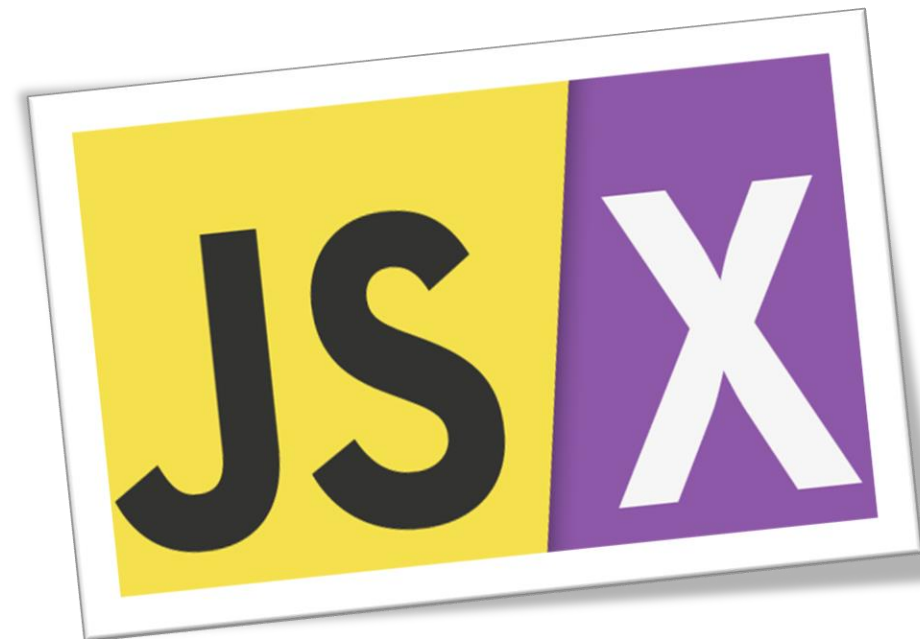
```
const h1Tag = <h1>Hola mundo, soy {nombre}</h1>;
```

En JS puro, llamado *Vanilla JS*, e intentando ponerlo en una sola línea equivaldría a:

```
const h1Tag = document.createElement('h1', null, `Hola mundo, soy ${nombre}`);
```

React puede trabajar sin JSX, pero con JSX el código es más limpio y más claro de leer, es una forma natural de utilizar el lenguaje HTML para trabajar con la interfaz de usuario dentro del código JS.

Y también permite que React muestre mensajes de error o advertencia más útiles.



*Vanilla JS* es el nombre con un poco de humor que le ha dado la comunidad al JavaScript puro, sin ninguna librería o *frameworks* de JS.





- ❑ JSX es estricto, por lo que toda etiqueta de apertura tiene que tener una etiqueta de cierre:

```
<h1>Hola mundo</h1>;
```

o

```
<input />;
```

- ❑ Cada componente de React debe tener un *return* y solo debe haber un elemento en el nivel más alto, no se puede devolver más de un elemento si no están anidados. Para ello, puede utilizarse un *fragment*, una etiqueta: `<> </>` que no renderiza nada, pero permite agrupar los diferentes elementos que se devuelven.

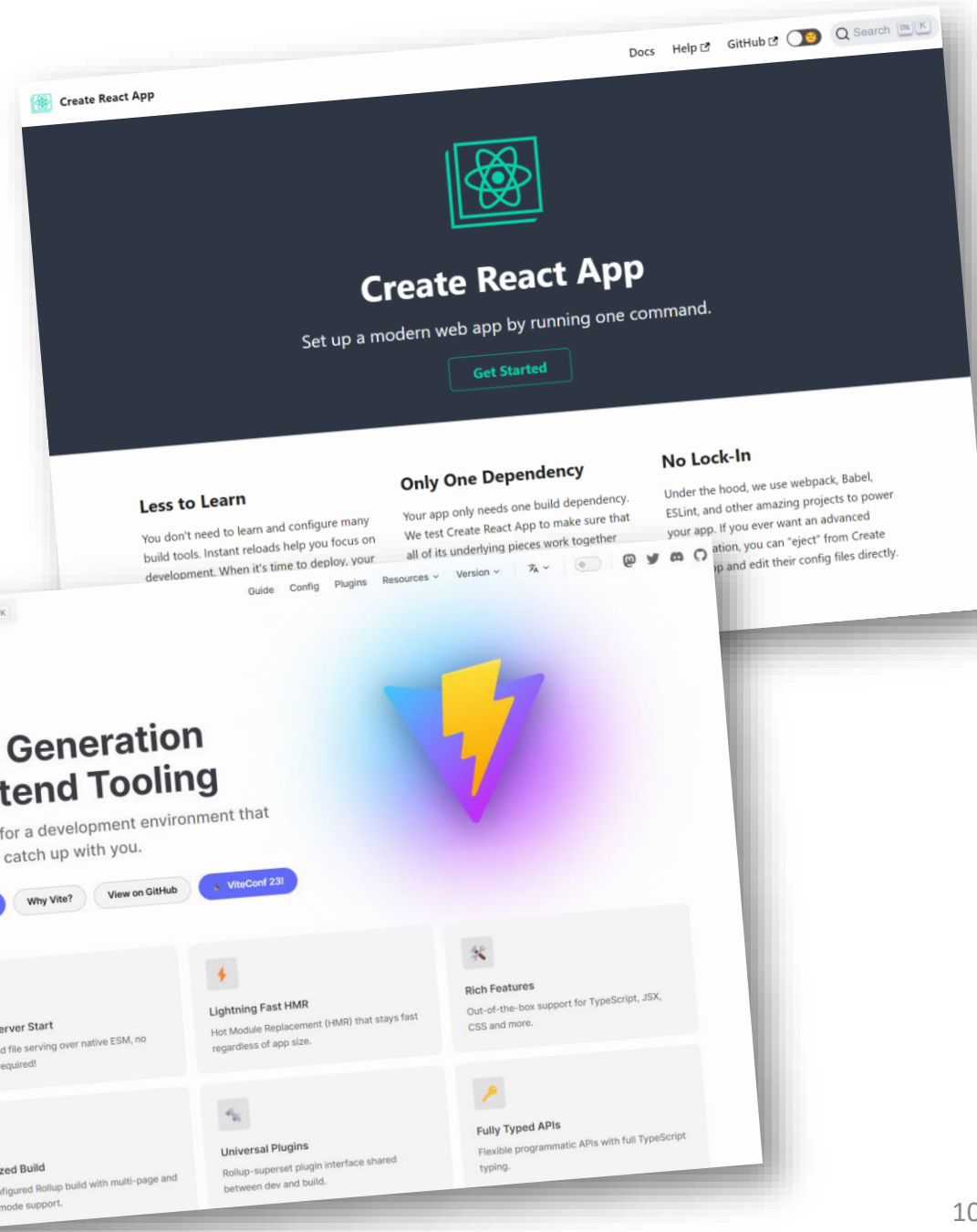
A large graphic of the letters 'JSX'. The 'JS' is in a bold, dark grey font on a yellow background. The 'X' is in a bold, white font on a purple background. The yellow and purple backgrounds are separated by a diagonal line.

# React App

Para crear un proyecto de React se utiliza Node.JS con los paquetes y configuración adecuados para trabajar con React.

Normalmente, no se construye desde cero, se utiliza una plantilla, una aplicación sencilla de React ya preconfigurada, como esqueleto o *framework*, con los paquetes y estructura de directorios ya preparados. Existen varias alternativas principales:

- ☐ *Create React App*, ya en desuso.
- ☐ *Vite*, en la actualidad es más optimo.
- ☐ Next.js
- ☐ Remis Run.
- ☐ Astro.



# Create React App

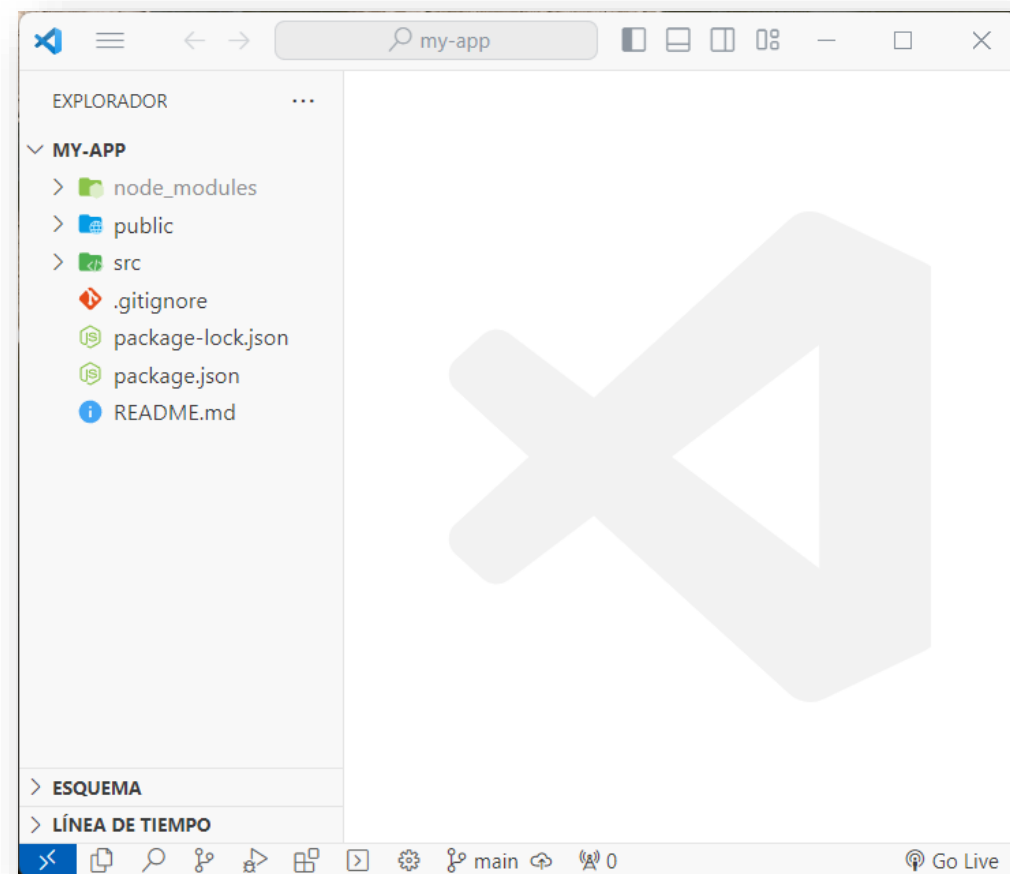
*Create React App* es un paquete de *NodeJS* que se instala de manera global para crear aplicaciones en React, lleva ya un año sin actualizarse. Internamente usa *Babel* y *Webpack*.

Su sitio web es: <https://create-react-app.dev/>

*Create React App* no trabaja con la lógica de *Backend* o de bases de datos, tan solo crea un flujo de construcción para *Frontend*, de manera que se puede usar con cualquier *Backend*.

Para crear una aplicación de React, en este caso llamada *my-app*, se ejecuta la siguiente instrucción desde el *prompt* del sistema:

Esto crea una carpeta llamada *my-app* que contiene la estructura de directorios y ficheros para trabajar en la aplicación de React.



```
npx create-react-app my-app
```

# [Create React App]

*npx* puede solicitar que se instale el paquete *create-react-app* para poder proceder con la creación de la aplicación:

```
PS C:\Users\ibm_h\Desktop> npx create-react-app hola-mundo
Need to install the following packages:
create-react-app@5.0.1
Ok to proceed? (y)
```

A continuación, procederá la instalación de forma normal:

```
PS C:\Users\ibm_h\Desktop> npx create-react-app my-app
Creating a new React app in C:\Users\ibm_h\Desktop\my-app.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
[Progress bar] | idealTree:@babel/core: sill placeDep ROOT semver@7.5.4 OK for: react-scripts@5.0.1 want: ^7.3.5
```

```
To address all issues (including breaking changes), run:
npm audit fix --force
```

```
Run 'npm audit' for details.
```

```
Created git commit.
```

```
Success! Created hola-mundo at C:\Users\ibm_h\Desktop\hola-mundo
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 hola-mundo
```

```
npm start
```

```
Happy hacking!
```

```
PS C:\Users\ibm_h\Desktop>
```

# 〔Create React App〕

La aplicación *my-app*, creada con *Create React App*, contiene el archivo *package.json* que indica que ha sido creada con *npm* (realmente con *npx*, pero en el tema anterior se vio que eran dos herramientas de la misma *suite*).

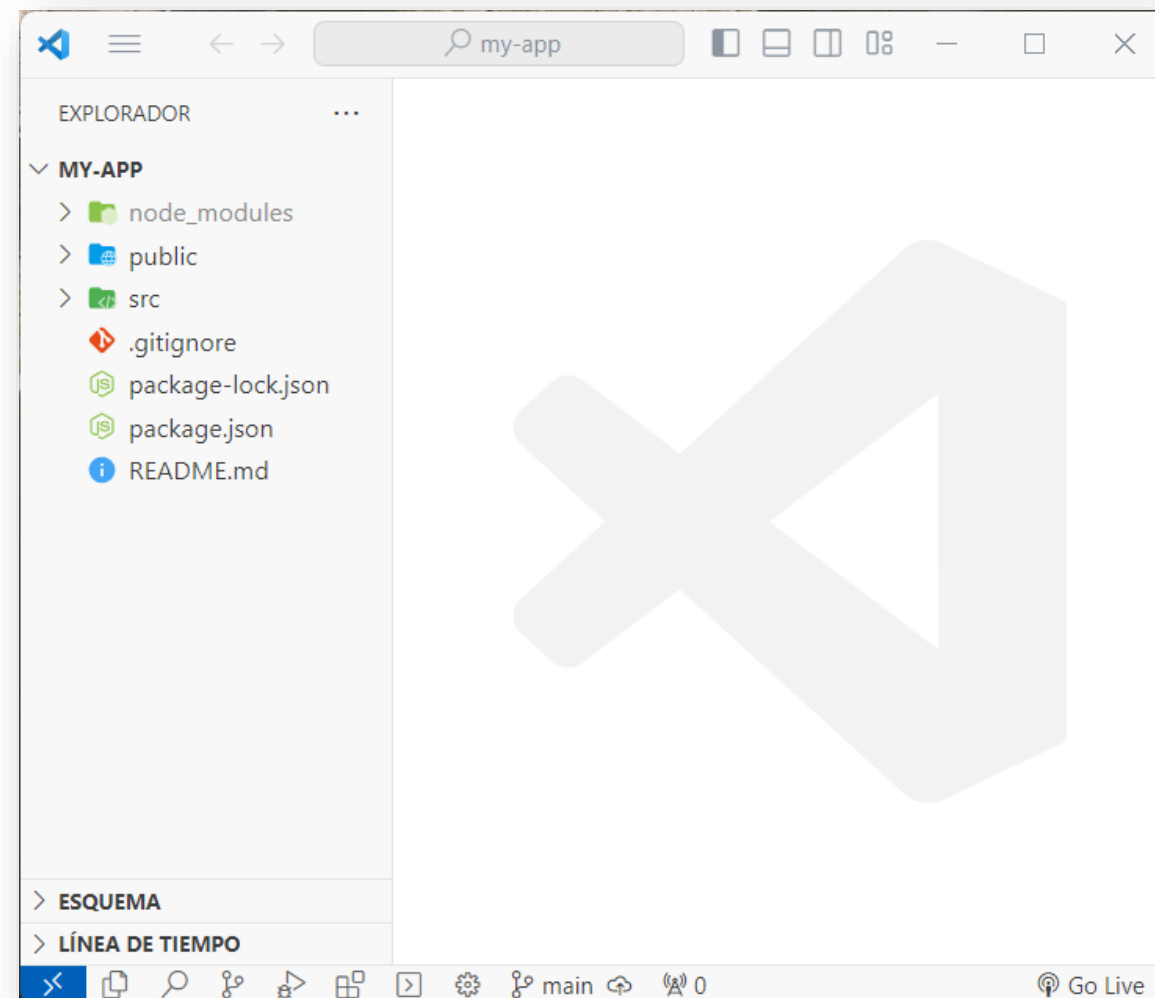
También se puede observar que el proyecto cuenta con soporte de *git*, por la presencia del archivo *.gitignore*.

Se puede ejecutar un *git log* y se observará que se ha realizado un *commit* de manera automática:

```
PS C:\Users\ibm_h\Desktop\my-app> git log
commit a08d37d9443abf91268d2e70b446512e5840e241 (HEAD -> main)
Author: Jorge García Flores <jgarciafl@educa.jcyl.es>
Date:   Sun Sep 24 11:52:11 2023 +0200
```

Initialize project using Create React App

```
PS C:\Users\ibm_h\Desktop\my-app> 
```





# [Create React App]

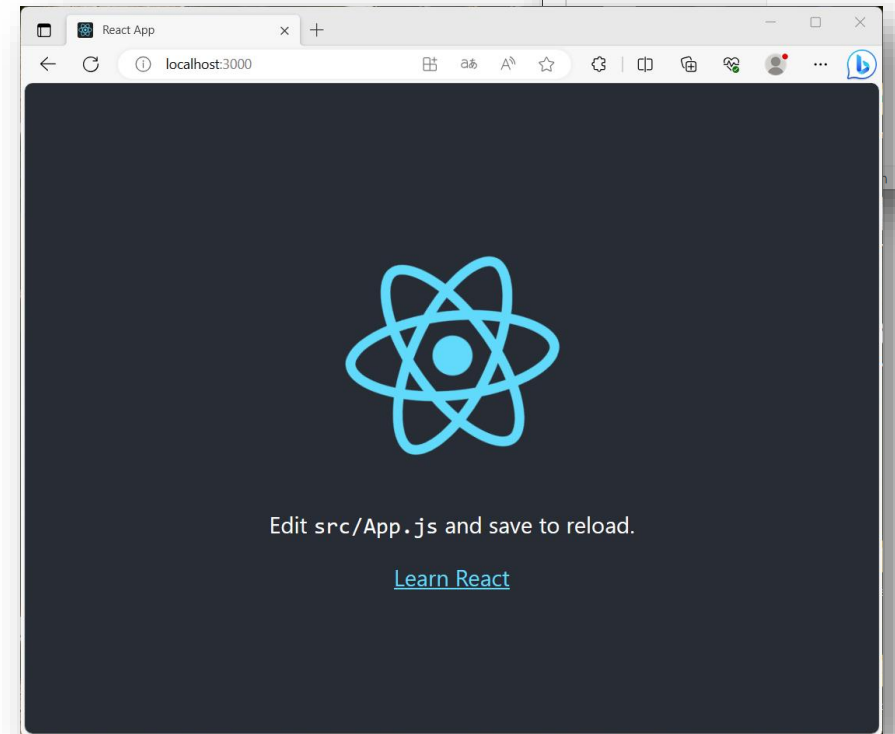
Realmente es una aplicación de Node.js, pero tiene la configuración para poder trabajar como una aplicación de React.

Esta es la forma en la que trabajan otros *frameworks* de JS como *Angular*, *Vue.JS*... Son aplicaciones de Node.js preconfiguradas, junto con los paquetes y dependencias, para trabajar con un *framework* particular.

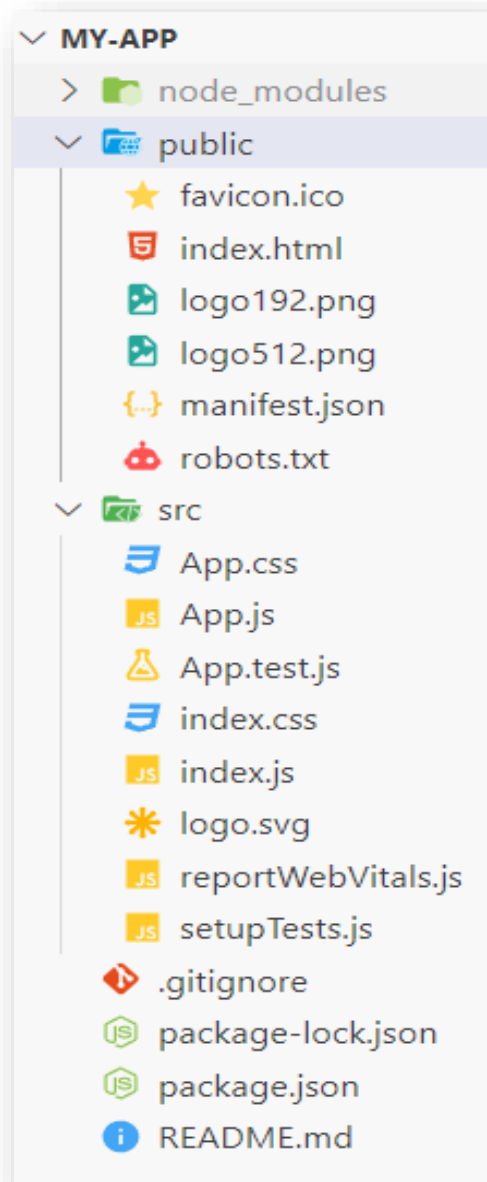
Como puede observarse, en la carpeta *node\_modules* están los paquetes y dependencias que necesita la aplicación, la mayoría de ellos son útiles para el desarrollo y no en producción.

Para lanzar la aplicación *my-app* se ejecuta el comando:

```
npm start
```



# Create React App



*Create React App* habrá creado tres carpetas:

- ❑ *node\_modules* que contiene los módulos y dependencias que necesita la aplicación.
- ❑ *public* que contiene las imágenes y archivos externos, como, por ejemplo, el *manifest.json* que sirve para crear la aplicación *PWA*, las imágenes que se usarán como icono de la aplicación *PWA*, el *favicon.ico* y la página *index.html* que cargará el script de JS que sirve como punto de entrada.
- ❑ *src* que contiene la aplicación de React propiamente dicha, mediante los ficheros *index.js* y *App.js*. Dentro de esta carpeta, además, se encuentran los archivos CSS, y se habrá generado el entorno de pruebas para utilizarlo con el paquete *JEST*.



Otra manera de crear una aplicación de *React*, sin partir de cero, es mediante *vite* para poder proceder con la creación de la aplicación. Se tiene un mayor control de la aplicación ya que solo instala las dependencias mínimas, por lo que si más adelante se necesitas otras características como, por ejemplo, routing o jest hay que instalar los paquetes manualmente.

Primero se necesita instalar el paquete de *vite*, *create-vite*, ya que no viene incluido con Node.js. Se instala el paquete de forma global, mediante *npm*, ejecutando la siguiente instrucción desde el *prompt* del sistema:

```
npm install vite
```

También se puede realizar directamente la instalación y creación de la aplicación React mediante cualquiera de los comandos:

```
npm init vite
```

```
npm create vite
```

Utilizando *yarn* primero se instalaría el paquete *yarn* y, a continuación, se crearía la aplicación:

```
npm i yarn
```

```
yarn create vite
```



*vite* preguntará las características que se necesitan para crear de la aplicación, como, por ejemplo, el nombre del proyecto, framework y la tecnología utilizada.

Hay que escoger en framework React (no Preact que es un React ligero).

En cuanto a la tecnología utilizada puede escogerse entre TypeScript y Vanilla JS, junto con SWC.

SWC (*Speedy Web Compiler* o Compilador Web Rápido) es la alternativa actual a *Babel*. Genera a partir de cualquier código JavaScript o TypeScript código JavaScript que funciona en navegadores antiguos y en modernos.

SWC es 20 veces más rápido que Babel ya que está escrito en el lenguaje optimizado para esta tarea, Rust.

```
C:\Users\ibm_h\Desktop>yarn create vite
yarn create v1.22.19
warning package.json: No license field
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Installed "create-vite@4.4.1" with binaries:
  - create-vite
  - cva
✓ Project name: ... my-app
? Select a framework: » - Use arrow-keys. Return to submit.
  Vanilla
  Vue
>  React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others
```



Una vez creada la estructura de directorios y ficheros de la aplicación React, desde la carpeta que contiene la aplicación, para que se satisfagan las dependencias de los paquetes, se ejecuta el comando:

```
yarn dev
```

Este comando creará la carpeta *node\_modules* y dentro de ella estarán los diferentes paquetes que se requieren.

A diferencia de *Create React App*, *yarn* instala menos paquetes, por lo que, si se requiere funcionalidad añadida, como, por ejemplo, JEST, para crear y ejecutar pruebas, habrá que instalarlo posteriormente de manera explícita.

```
C:\Users\ibm_h\Desktop>yarn create vite
yarn create v1.22.19
warning package.json: No license field
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Installed "create-vite@4.4.1" with binaries:
  - create-vite
  - cva
✓ Project name: ... my-app
✓ Select a framework: » React
✓ Select a variant: » JavaScript

Scaffolding project in C:\Users\ibm_h\Desktop\my-app...

Done. Now run:

  cd my-app
  yarn
  yarn dev

Done in 148.92s.
```





La aplicación *my-app*, creada con *vite*, contiene un archivo *package.json* por lo que si se quisiera trabajar con *npm* (en lugar de con *yarn* bastaría con borrar, si existiera, el archivo *yarn.Log* y la carpeta *node\_modules*, y a continuación hacer un *npm install*.

También se puede observar que el proyecto cuenta con soporte de *git*, por la presencia del archivo *.gitignore*, pero a diferencia de *Create React App*, no se ha inicializado *git* y tampoco se ha realizado ningún *commit*:

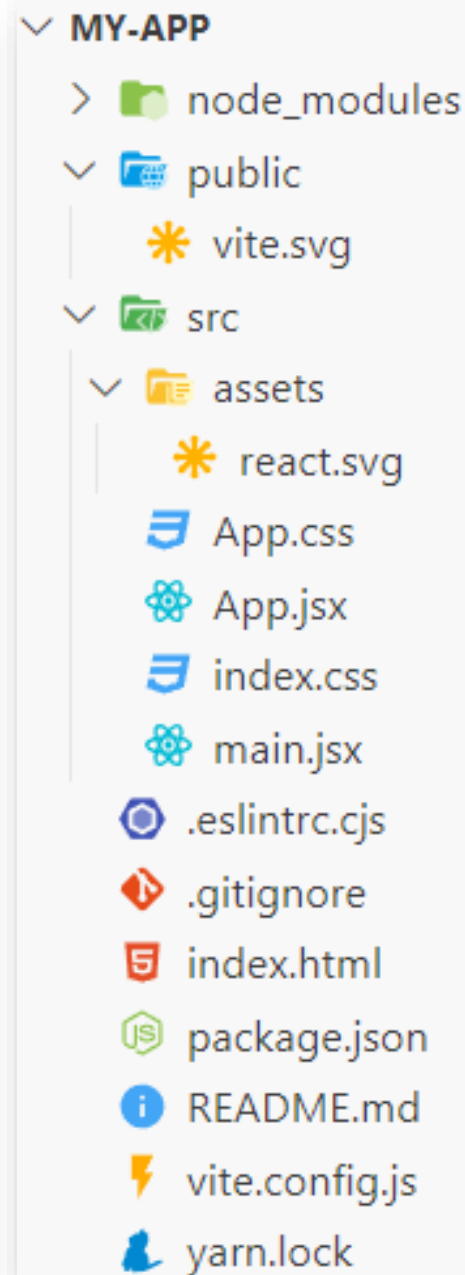
```
PS C:\Users\ibm_h\Desktop\my-app> git log
fatal: not a git repository (or any of the parent directories): .git
PS C:\Users\ibm_h\Desktop\my-app>
```

En el archivo *README.md* presenta información de *React + Vite*:

This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.

Currently, two official plugins are available:

- [@vitejs/plugin-react](#) uses [Babel](#) for Fast Refresh
- [@vitejs/plugin-react-swc](#) uses [SWC](#) for Fast Refresh



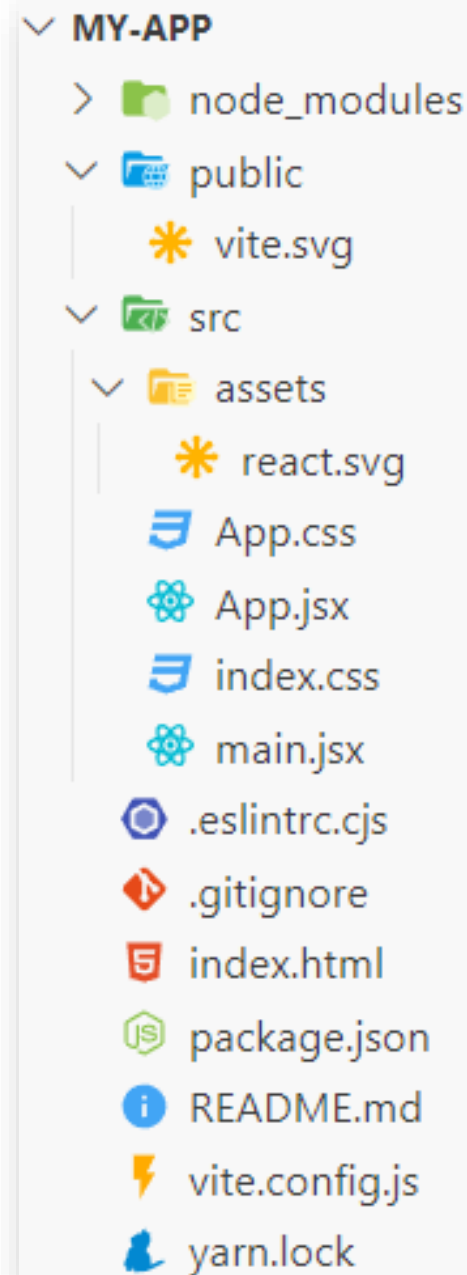


Básicamente expone los dos modos de funcionamiento de Vite para trabajar con React, usar Babel o SWC.

En el fichero de configuración de Vite es *vite.config.js* y contiene el modo de funcionamiento escogido:

```
⚡ vite.config.js > ...  
1  import { defineConfig } from 'vite'  
2  import react from '@vitejs/plugin-react'  
3  
4  // https://vitejs.dev/config/  
5  export default defineConfig({  
6    | plugins: [react()],  
7  })
```

Se aprecia en la imagen anterior que se utiliza en la configuración actual del proyecto que se va a trabajar con Babel ya que se usa el plugin *@vitejs/plugin-react*. El resto de la configuración puede consultarse en <https://vitejs.dev/config/>.

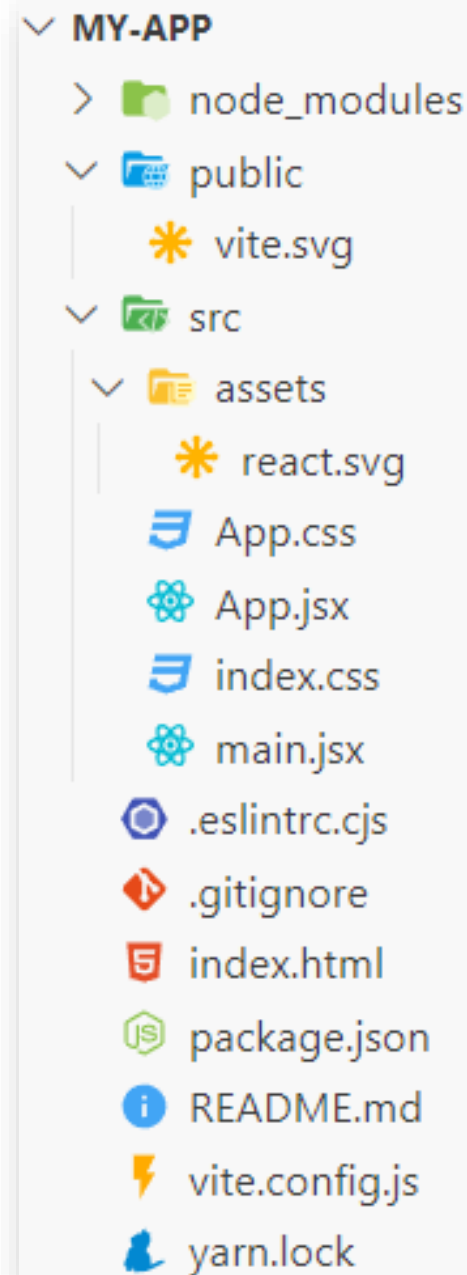




Vite habrá creado también tres carpetas:

- ❑ *node\_modules* que contiene los módulos y dependencias que necesita la aplicación.
- ❑ *public* que contiene el icono de vite, *vite.svg*.
- ❑ *src* contiene la aplicación de React propiamente dicha, y a su vez contiene otra carpeta llamada *assets*, que sirve para almacenar imágenes, iconos... En *src* se encuentran los ficheros *main.js* (que representa al fichero *index.js* de *Create React App*) y *App.jsx*., junto con los ficheros CSS asociados: *index.css* y *App.css*.

Por último, en la raíz del proyecto hay un fichero llamado *.eslintrc.js* que es un fichero de configuración de *ESLint*, un *linter* muy utilizado en JS. Este fichero contiene las reglas para utilizar React correctamente la aplicación, advirtiendo y corrigiendo los posibles problemas que puedan existir.





Para lanzar la aplicación React creada con vite se consulta el fichero *package.json*, como cualquier otra aplicación de *Node.js*, en la sección de *scripts*:

```
"scripts": {  
  "dev": "vite",  
  "build": "vite build",  
  "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",  
  "preview": "vite preview"  
},
```

- ❑ *dev* se utiliza para lanzar la aplicación en modo desarrollo (sin hacer el *treeshaking* ni ningún otro proceso parecido).
- ❑ *build* genera la aplicación para producción, creando una carpeta *dist* que puede llevarse al servidor.
- ❑ *lint* ejecuta en *linter* para comprobar los posibles errores y así poder corregirlos.
- ❑ *preview*



Esos comandos se ejecutan con los comandos *npm* o con *yarn*, dependiendo del gestor de paquetes que se esté utilizando, se recomienda utilizar uno de los dos y no ir cambiando del uno al otro.

Por ejemplo, para lanzar el *linter*:

```
yarn lint
```

Produce la siguiente salida:

```
PS C:\Users\ibm_h\Desktop\my-app> yarn lint
yarn run v1.22.19
warning ..\..\package.json: No license field
$ eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0

C:\Users\ibm_h\Desktop\my-app\src\App.jsx
  12:9  error  Using target="_blank" without rel="noopener" (which implies rel="noopener") is a security risk in older browsers: see https://mathiasbynens.github.io/rel-noopener/#recommendations  react/jsx-no-target-blank
  15:9  error  Using target="_blank" without rel="noopener" (which implies rel="noopener") is a security risk in older browsers: see https://mathiasbynens.github.io/rel-noopener/#recommendations  react/jsx-no-target-blank

X 2 problems (2 errors, 0 warnings)
2 errors and 0 warnings potentially fixable with the `--fix` option.

error Command failed with exit code 1.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
```

Se puede solucionar con la siguiente ejecución:

```
yarn lint --fix
```





Para lanzar la aplicación en modo desarrollo:

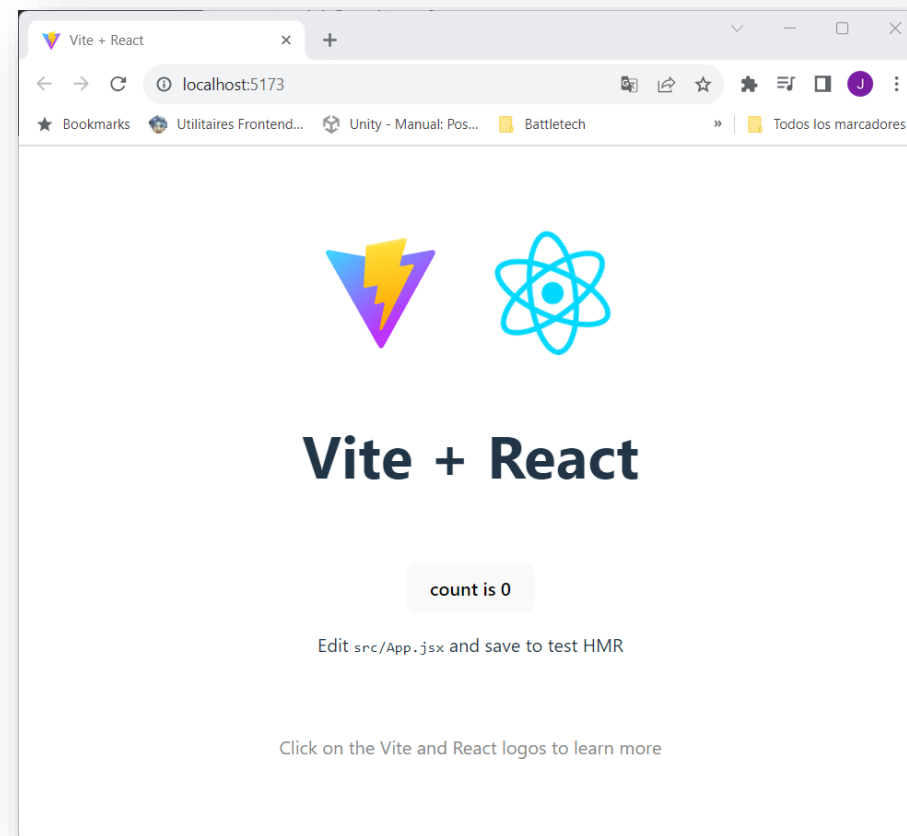
`yarn dev`

Produce la siguiente salida:

```
VITE v4.4.10 ready in 425 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h to show help

Shortcuts
press r to restart the server
press u to show server url
press o to open in browser
press c to clear console
press q to quit
```



Esto carga la aplicación en la siguiente dirección <http://localhost:5173/>

Para salir de la aplicación se pulsa la tecla `q`



Para generar la aplicación de producción se ejecuta el siguiente comando:

```
yarn build
```

Esto crea una carpeta llamada *dist* que contiene toda la aplicación, pero por defecto debe estar en la raíz del servidor. Para que la aplicación se pueda desplegar en una carpeta del servidor, y que no fallen las rutas a los diferentes elementos, como las rutas a las imágenes, scripts de JS, etc., hay que modificar el archivo de configuración de Vite *vite.config.js*:

Por ejemplo, para desplegar la aplicación, cuando se construye la versión de producción, en la carpeta llamada *nombre-carpeta* del servidor web hay que configurar el fichero *vite.config.js* de la siguiente manera:

```
export default defineConfig(({ command }) => ({  
  base: command === 'build' ? `/nombre-carpeta/` : '/',  
  plugins: [react()],  
}));
```

# Devops con vite

Para que se despliegue en el servidor <http://143.47.43.204:8080> dentro de la carpeta de un usuario llamado *jorge* y, dentro de esta carpeta, en otra llamada como el nombre del proyecto asignado en *GitLab* sería:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { getGitProjectName } from './miConfig'

const usuario='jorge';

export default defineConfig(({ command }) => ({
  base: command === 'build' ? `/${usuario}/${getGitProjectName()}/` : '/',
  build: {
    rollupOptions: {
      output: {
        assetFileNames: 'resources/[ext]/[name][extname]',
        chunkFileNames: 'resources/chunks/[name].[hash].js',
        entryFileNames: 'resources/js/[name].js',
      },
    },
    plugins: [react()],
  })});
```

# Devops con vite

Como se aprecia en la configuración anterior es necesario crear el fichero *miConfig.js* con la función *getGitProjectName* en la misma carpeta donde se encuentre el fichero *vite.config.js*:

```
import { execSync } from 'child_process';

const defaultProjectName="dist";

export function getGitProjectName() {
  try {
    const remoteUrl = execSync(`git config --get remote.origin.url`).toString().trim();
    const projectName = remoteUrl.split('/').pop().replace('.git', '');
    return projectName;
  } catch (error) {
    try {
      const branch = execSync('git branch --show-current').toString().trim();
      const remoteUrl = execSync(`git config --get branch.${branch}.remote`).toString().trim();
      const projectName = remoteUrl.split('/').pop().replace('.git', '');
      return projectName;
    } catch (error) {
      console.error('Error al obtener el nombre del proyecto desde Git:', error);
      return defaultProjectName; // Nombre por defecto si no se puede obtener desde Git
    }
  }
}
```

# [Devops con vite]

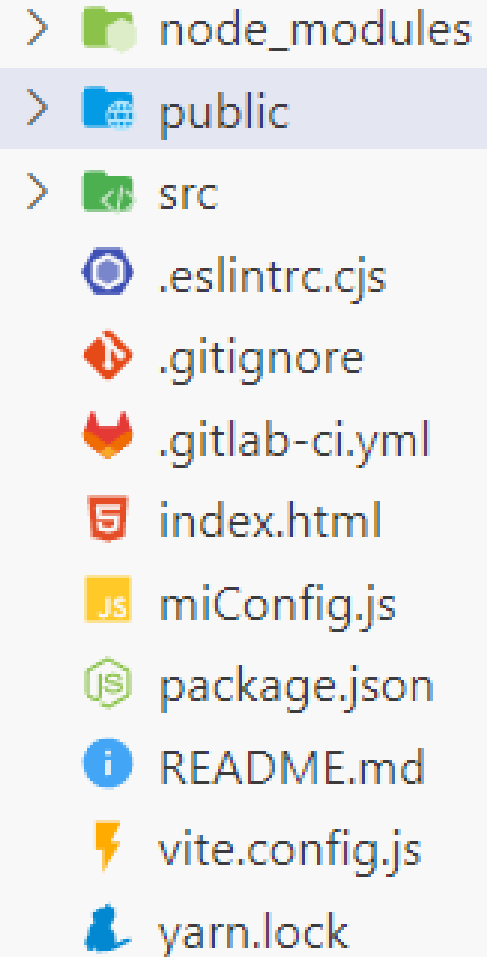
La función *getGitProjectName* devuelve el nombre del proyecto asignado en *GitLab* partir de la configuración de *git*, si no encuentra la configuración le asigna por defecto el nombre de *dist*.

De igual manera para que pueda desplegarse mediante operaciones *CI/CD* de *GitLab* hay que comentar la siguiente línea del fichero *.gitignore*, anteponiendo *#* a la palabra *dist*:

```
#dist
```

La estructura de directorios queda de la siguiente manera:

A continuación, se realiza el *yarn build*.



A file explorer window showing the project structure. The 'public' directory is selected and highlighted in blue. The structure is as follows:

- > node\_modules
- > public
- > src
  - .eslintrc.cjs
  - .gitignore
  - .gitlab-ci.yml
  - index.html
  - miConfig.js
  - package.json
  - README.md
  - vite.config.js
  - yarn.lock



# (Devops con vite)

Se configura el fichero `.gitlab-ci.yml` para que permita hacer el despliegue de forma automática:

```
stages:
  - deploy

deploy-job:
  stage: deploy
  environment: production
  script:
    - echo "Desplegando la aplicación..."
    - lftp -c "open $FTP_SERVER; user $FTP_USERNAME $FTP_PASSWORD; mkdir -p $CI_PROJECT_NAME; lcd dist;
mirror --ignore-time -Rev --parallel=10 --include ./*** --exclude README.md --exclude public/ --exclude
src/ --exclude .eslintrc.cjs --exclude miConfig.js --exclude package.json --exclude vite.config.js --
exclude yarn.lock --exclude .gitlab-ci.yml --exclude .git/ --exclude Readme-Imagenes/ --exclude
.gitignore ./ ./ $CI_PROJECT_NAME; bye"
    - echo "Aplicación exitosamente desplegada."
    - echo "Disponible en http://${FTP_SERVER}:8080/${FTP_USERNAME}/${CI_PROJECT_NAME}."
```

# [Devops con vite]

Se inicializa el proyecto con *git*:

```
git init --initial-branch=main  
git add .  
git commit -m "Primer commit"
```

Se escoge el nombre del proyecto para *GitLab*, en este caso, el proyecto *hola-vite.git*, dentro del subgrupo *react*, del grupo *dwec*:

```
git push --set-upstream git@143.47.43.204:dwec/react/hola-vite.git main
```

Por último, se vuelve a generar la versión de producción para que se generen las rutas correctamente, se realiza un *commit* con los cambios y se envían al servidor *GitLab* para almacenar el proyecto en el repositorio y para que se despliegue correctamente:

```
yarn build  
git add .  
git commit -m "Configuración de despliegue"  
git push
```

# Extensiones VSCode de React

- ❑ ES7+ React/Redux/React-Native snippets:

<https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets>

- ❑ Simple React Snippets:

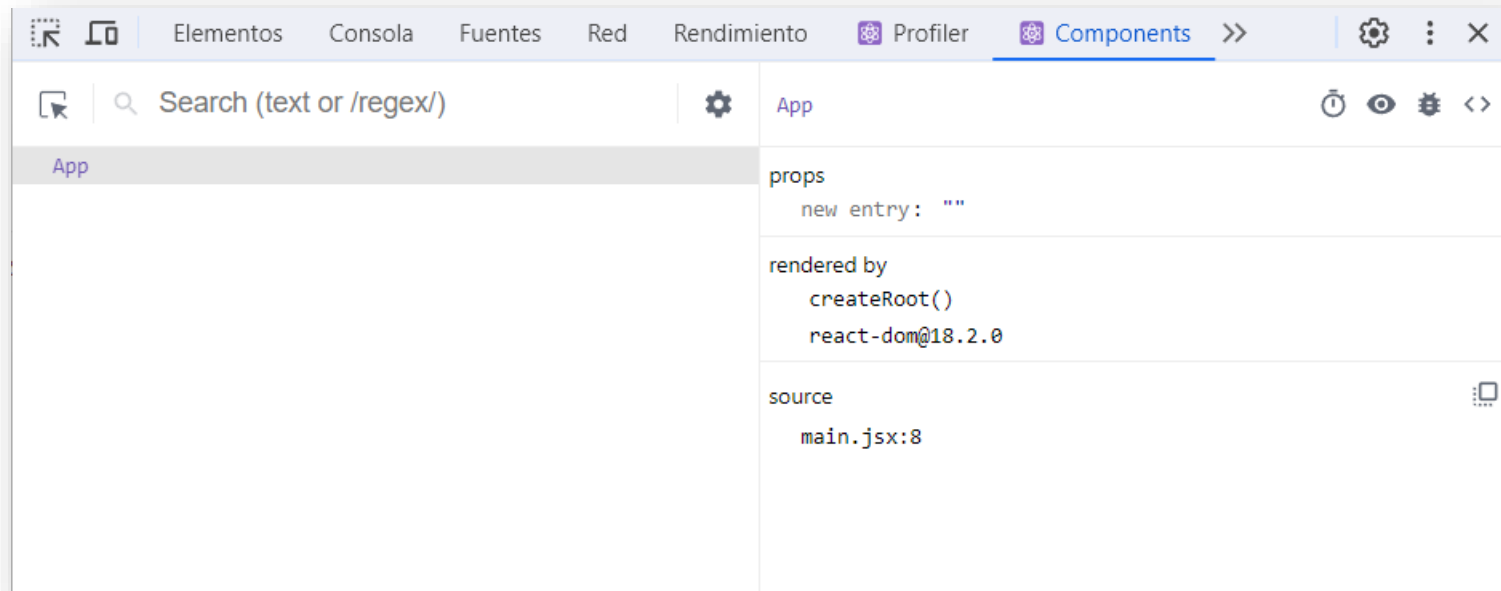
<https://marketplace.visualstudio.com/items?itemName=burkeholland.simple-react-snippets>

# Extensiones para el navegador

## ❑ React Developer Tools

Es una herramienta que permite visualizar e inspeccionar los componentes de React en el navegador Google y Firefox.

[https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?utm\\_source=ext\\_sidebar&hl=es](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?utm_source=ext_sidebar&hl=es)



# Bibliografía y recursos online

- <https://es.react.dev/>
- <https://matiashernandez.dev/blog/post/que-es-jsx-por-que-usamos-jsx-en-react-y-como-funciona>
- <https://create-react-app.dev/>
- <https://vitejs.dev>
- <https://vitejs.dev/config/>
- <https://eslint.org/>