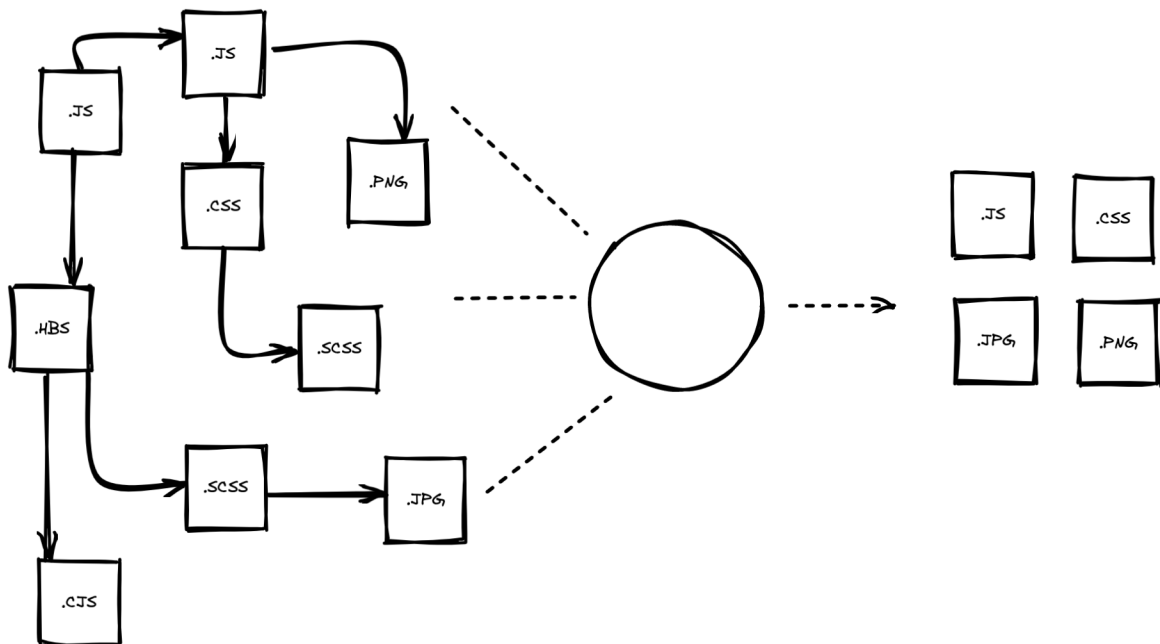


Vite Bundler

Cualquier aplicación web “tradicional” esta compuesta por **HTML**, **CSS** y **JavaScript**. Estos 3 elementos combinados darán forma, estilo y funcionalidad a nuestra aplicación web.

Llegados a este punto deberíamos de encontrar la forma más adecuada de tener un entorno de desarrollo lo más parecido posible a lo que vamos a tener en el módulo de **React**, haciendo uso además de **librerías** o **bibliotecas** externas que nos compilen al formato anteriormente mencionado.



Para ello haremos uso de los llamados “**bundlers**”, que son los encargados de recopilar en un mismo lugar las librerías, nuestros archivos HTML, CSS y JavaScript propios, compilarlo y devolverlo en un formato unificado al navegador.

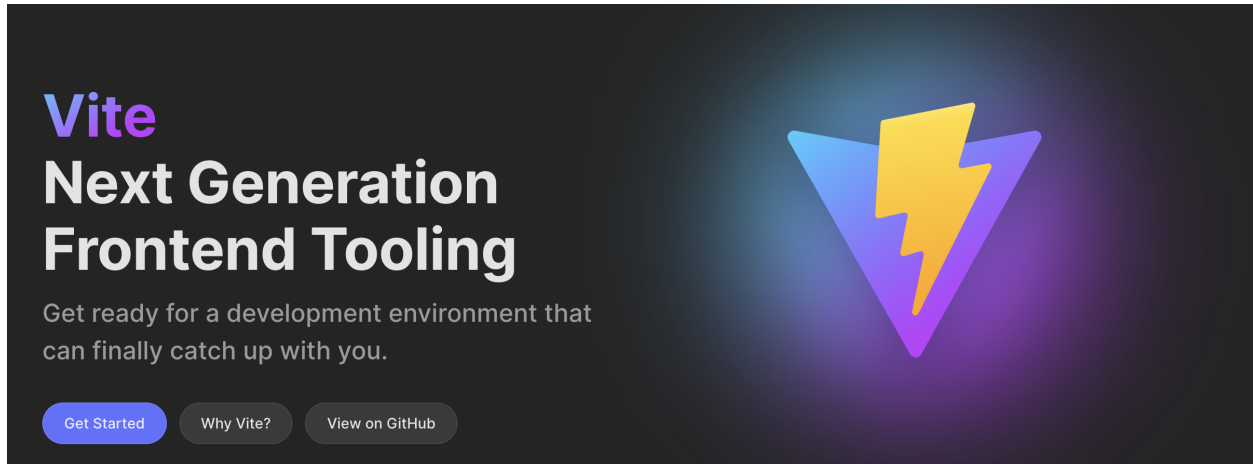
El bundler en cuestión que vamos a utilizar se llama **Rollup**, y lo haremos a través de **Vite**.

Qué es Vite?

Vite es un conjunto de herramientas para el front-end en el que utilizaremos por debajo un bundler llamado **Rollup** y toda la configuración que Vite nos aporta para tener un entorno de desarrollo rápido en el que disfrutaremos de varias características:

- **Hot reloading**: al cambiar cualquiera de los archivos de nuestra aplicación, Vite volverá a recargarla con los cambios reflejados para que podamos trabajar en “tiempo real” en el navegador.+
- **Servidor de desarrollo instantáneo**.

- **Builds** finales optimizadas para publicación a través de Rollup.
- **Optimización** de CSS.



En el siguiente bloque generaremos nuestro propio **HTML**, **CSS** y **JavaScript** tal y como hemos hecho en anteriores ocasiones, y a través de este bundler generaremos una versión de nuestra aplicación web más optimizada combinando nuestro código con librerías externas.

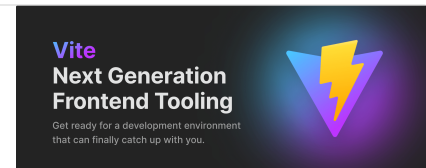
Probando Rollup & Vite

Vamos a crear nuestro primer proyecto con **Vite**. Para ello vamos a acceder a la página web oficial de Vite dentro de la sección de **Guía**, la cual nos indicará paso a paso cómo podemos arrancar un proyecto desde nuestro entorno de desarrollo.

Vite

Vite (palabra en francés para "rápido", pronunciado como /vit/, como "veet") es una herramienta de compilación que tiene como objetivo proporcionar una experiencia de desarrollo más rápida y ágil para proyectos web modernos.

<https://es.vitejs.dev/guide/>



Dentro de esta guía comprobaremos los diferentes templates que nos ofrece Vite, los cuales nos permitirán trabajar con diferentes configuraciones de lenguaje y tecnologías con las que trabajar en nuestro proyecto.

Probar Vite online

Puedes probar Vite online en [StackBlitz](#). Éste ejecuta la configuración de compilación basada en Vite directamente en el navegador, por lo que es casi idéntica a la configuración local pero con la diferencia que no requiere que instales nada en tu máquina. Puedes navegar a `vite.new/{template}` para seleccionar qué marco de trabajo utilizar.

Los ajustes preestablecidos de plantilla admitidos son:

JavaScript	TypeScript
vanilla	vanilla-ts
vue	vue-ts
react	react-ts
preact	preact-ts
lit	lit-ts
svelte	svelte-ts

En nuestro caso vamos a probarlo con **JavaScript vanilla**, ya que es el lenguaje con el que hemos estado trabajando hasta ahora. Para ello vamos a seguir los comandos que nos indica el paso a paso contemplado por la guía para crear nuestro proyecto con Vite y la configuración seleccionada.

Lo primero que tendremos que ejecutar es la instalación del paquete principal a través de **NPM** mediante el siguiente comando:

```
npm create vite@latest
```

Este comando desencadenará una serie de preguntas y opciones en nuestra terminal que podremos seleccionar a través de las flechas del teclado e introduciendo la información requerida.

La primera de ellas, en el caso de que sea la primera vez que utilizamos Vite, será si queremos instalar de manera global la librería de Vite. En esta opción marcaremos una **S** o **Y** para proceder a la instalación.

Lo siguiente que nos preguntará la terminal es el nombre de nuestro proyecto, en nuestro caso lo llamaremos ejemplo-vite.

```
? Project name: > ejemplo-vite
```

El último paso será indicar a través de las flechas del teclado el framework o stack que utilizaremos en nuestro proyecto. En esta ocasión vamos a trabajar con **vanilla** y su variante **vanilla (sin TypeScript)**.

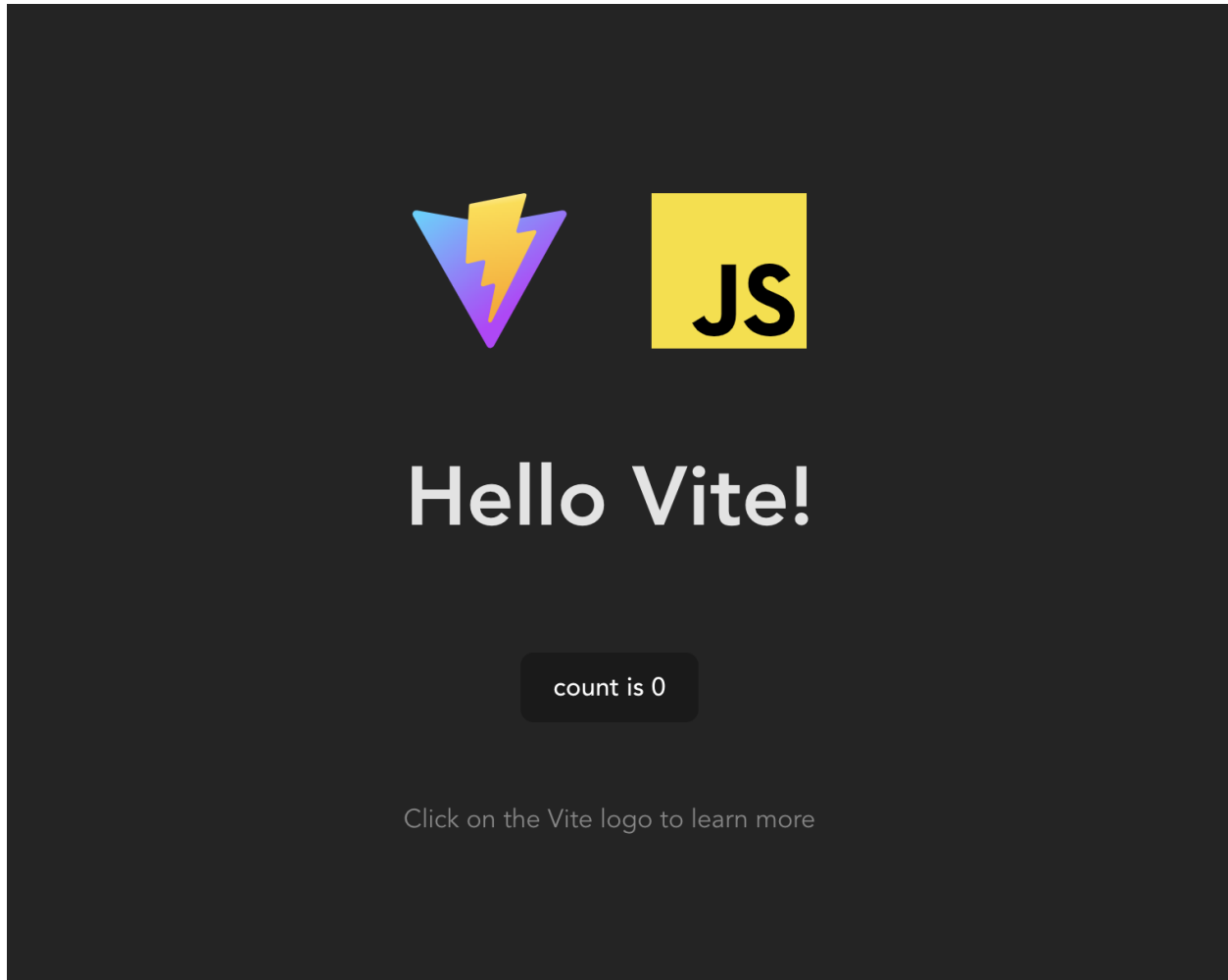
```
✓ Project name: ... ejemplo-vite
✓ Select a framework: > vanilla
✓ Select a variant: > JavaScript
```

Tras estos comandos, la terminal nos devolverá una serie de instrucciones a seguir para acceder a la carpeta del proyecto, instalar las dependencias y arrancar la versión de desarrollo en el servidor:

```
cd ejemplo-vite //Accedemos al directorio creado
```

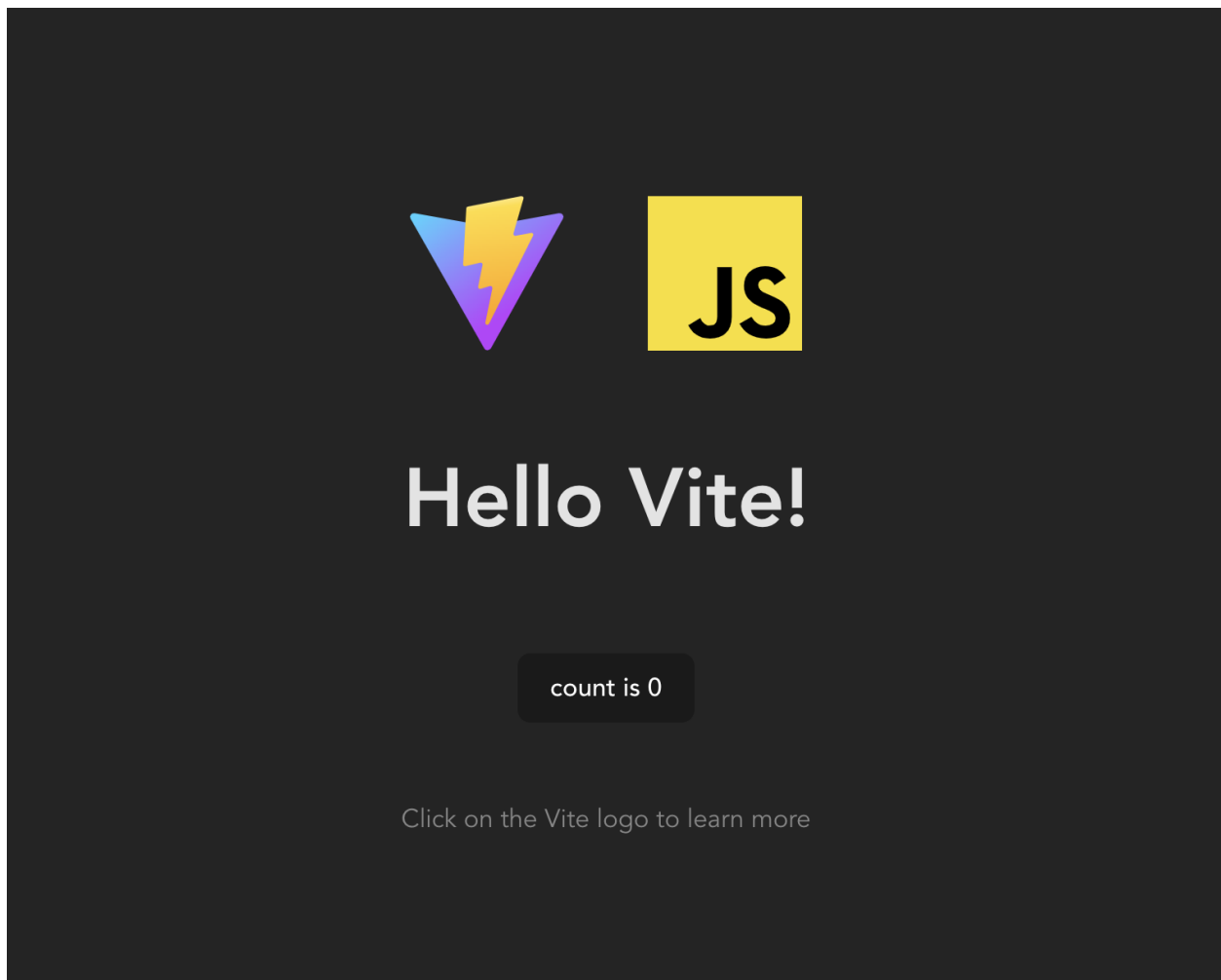
```
npm install    //Instalamos las dependencias del package.json
npm run dev    //Arrancamos el servidor
```

Completados estos pasos, la terminal nos devolverá un mensaje con la dirección del puerto local donde está corriendo nuestro proyecto en su versión de desarrollo, por lo cual, si accedemos al puerto <http://127.0.0.1:5173/> (en nuestro caso es este, pero dependiendo de la configuración de cada máquina puede variar), podremos ver el proyecto corriendo en nuestro navegador:



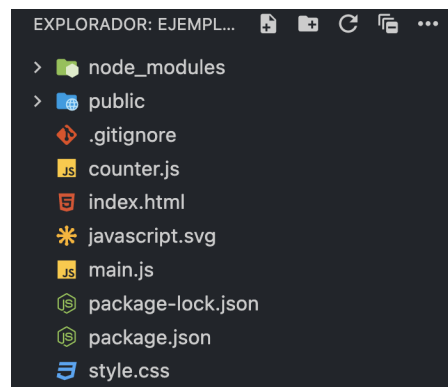
Estructura de ficheros

Una vez creado el proyecto y arrancado el servidor con la información por defecto que nos proporciona el template de **Vite** tendremos en el navegador una vista de nuestro proyecto en su versión de desarrollo:



En esta pantalla podremos ver un enlace a la documentación de **Vite**, a la documentación oficial de MDN acerca de **JavaScript** y un contador interactivo que nos cuenta los clicks que realizamos sobre él.

Vamos a inspeccionar en nuestro editor de código los ficheros y carpetas que están conformando nuestro proyecto.



- La carpeta **node_modules** contiene todas las librerías instaladas a través de NPM que estamos utilizando en el proyecto. Esta carpeta tiene mucho peso y se creará cada vez que ejecutemos **npm install** o instalemos cualquier dependencia.
- El fichero **package.json** definirá el “paquete” de nuestra aplicación, el cual contiene el nombre del proyecto, la privacidad, la versión del mismo, el tipo (en este caso es module porque es un modulo independiente), los scripts que podemos lanzar a través de la terminal y por último las dependencias instaladas. En este caso, Vite tenía en el package.json la dependencia de vite incluida por defecto, por lo que al haber realizado **npm install** en el último paso de la configuración del proyecto nos ha instalado dicha dependencia y generado la carpeta **node_modules** con la instalación de la misma.
- El fichero **.gitignore** contiene todos los archivos o carpetas que tendrán que ser ignorados por git. Por ejemplo, la carpeta node_modules debido a su gran tamaño.
- La carpeta **public** albergará todos los archivos o recursos que utilizaremos en nuestro proyecto.
- El resto de ficheros nos serán más familiares, ya que son scripts de **JavaScript**, un fichero **index.html** que enlaza la funcionalidad y un **style.css** con nuestra hoja de estilos.

En el fichero **index.html** se importa por defecto el archivo **main.js** y, además contiene un div con un id denominado **app**.

```
[...]
<div id="app"></div>
<script type="module" src="/main.js"></script>
[...]
```

Si inspeccionamos el archivo **main.js** podremos ver como está volcando a través de un script una serie de código con funcionalidad en dicho **div**, generando una serie de bloques en HTML y trayendo a su vez la funcionalidad de otro script denominado **counter.js**, que es la funcionalidad de el contador de clicks que hemos visto en el navegador.

```
import './style.css'
import javascriptLogo from './javascript.svg'
import { setupCounter } from './counter.js'

document.querySelector('#app').innerHTML = `
  <div>
    <a href="https://vitejs.dev" target="_blank">
      
    </a>
    <a href="https://developer.mozilla.org/en-US/docs/Web/JavaScript" target="_blank">
      
    </a>
    <h1>Hello Vite!</h1>
    <div class="card">
      <button id="counter" type="button"></button>
    </div>
    <p class="read-the-docs">
      Click on the Vite logo to learn more
    </p>
  </div>
`

setupCounter(document.querySelector('#counter'))
```

En fichero **main.js** está importando los estilos de **style.css** con unos dados por Vite. Además de esto está importando un **svg** con el logo de JavaScript que ha incluido en la raíz de nuestro proyecto y que podemos ver cómo está pintando mediante una etiqueta **img**.

También está importando la función **setUpCounter**, a la cual le está pasando el botón counter para modificar su **innerHTML**, inyectando toda la funcionalidad del **counter.js** en **main.js**. Como veis todo está bastante bien compartimentando y dividido según su funcionalidad.

Como podemos comprobar si hacemos cualquier cambio en nuestros archivos, al guardar alguno de estos cambios tardará milésimas de segundos en reflejárnoslos en el navegador. Esto es lo que hemos mencionado en la introducción como **Hot Reloading**. Con esta funcionalidad podremos desarrollar a “tiempo real” viendo cualquier cambio que hagamos en “caliente”.



Creando un Bundle

Pero, ¿dónde está la parte en la que **Rollup** recopila todos mis archivos y genera una versión optimizada de todo?

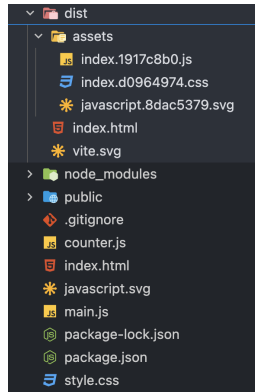
Para ello tendremos que ejecutar el último script reflejado en **package.json** denominado **build**, y que ejecutaremos mediante el siguiente comando:

```
npm run build
```

Al ejecutar este comando, Vite nos creará una carpeta llamada **dist** con una serie de archivos un tanto diferentes a los que teníamos originalmente.

```
> ejemplo-vite@0.0.0 build
> vite build

vite v3.0.4 building for production...
✓ 6 modules transformed.
dist/assets/javascript.8dac5379.svg 0.97 KiB
dist/index.html 0.44 KiB
dist/assets/index.d0964974.css 1.19 KiB / gzip: 0.62 KiB
dist/assets/index.1917c8b0.js 1.41 KiB / gzip: 0.74 KiB
```



Estos archivos contendrán una versión “minificada” de todos nuestros scripts y archivos. La denominación de los mismos con números y caracteres viene dada por Vite según la información interna de cada uno.

Si inspeccionamos dichos archivos podremos comprobar que lo que más llama la atención es el nuevo fichero **index.1917c8b0.js** que nos ha generado, con una versión optimizada, con namings mínimos y con la combinación más optima de nuestro fichero main y el fichero counter en un solo fichero.

```
const c=function(){const o=document.createElement("link").relList;if(o&&o.supports&&o.supports("modulepreload"))return;for(const e of docum
<div>
  <a href="https://vitejs.dev" target="_blank">
    
  </a>
  <a href="https://developer.mozilla.org/en-US/docs/Web/JavaScript" target="_blank">
    
  </a>
  <h1>Hello!</h1>
  <div class="card">
    <button id="counter" type="button"></button>
  </div>
  <p class="read-the-docs">
    Click on the Vite logo to learn more
  </p>
</div>
`;a(document.querySelector("#counter"));
```

Lo que ha hecho Vite es optimizar lo máximo posible todo lo que hemos implementado en diferentes ficheros para crear una versión más ligera, más optima y más simple para el navegador.

Nosotros siempre trabajaremos sobre la versión estándar de desarrollo, pero a la hora de publicar la página o desplegarla en algún servicio de despliegue lo que utilizaremos será esta versión que nos genera Vite. Bajo ningún concepto tendremos que tocar los archivos que nos ha generado el **build** en la carpeta **dist**.

Si queremos ver una previsualización de esta versión procesada generada a través del build podemos ejecutar el script **preview** que nos proporciona Vite, mediante el cual podremos ver las diferencias de rendimiento directamente en el navegador en un puerto diferente.

```
npm run preview
```

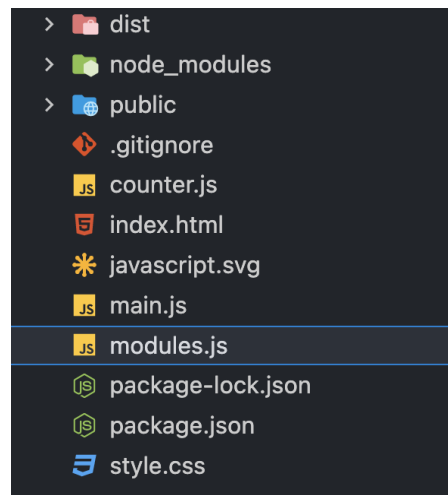
De esta forma estaremos viendo en el navegador la versión optimizada por Rollup y Vite de nuestra aplicación web.

Imports & Exports

Ahora que ya sabemos utilizar Vite para crear nuestros proyectos con el lenguaje que queramos, vamos a ver un concepto muy importante en proyectos con varios archivos que es **import** y **export**.

Hasta ahora hemos trabajado solamente con un fichero de JavaScript y hemos volcado su funcionalidad en un fichero HTML, pero como hemos visto en el proyecto del template de Vite podemos tener varios que aniden scripts según su funcionalidad (por ejemplo).

Gracias a Vite tenemos **módulos**, con los cuales tendremos nuestro código más organizado con funcionalidades exportadas e importadas desglosándolo en pequeñas partes y ficheros JavaScript. Para ello vamos a crear un nuevo archivo en la raíz de nuestro proyecto de ejemplo llamado **modules.js**:



Vamos a ver cómo funcionan los módulos a modo demostrativo.

Para ello vamos a empezar creando una nueva constante en nuestro fichero:

```
const person = {  
  name: "Stephen",  
  surname: "Strange"  
};
```

Esta constante existe dentro de nuestro archivo **modules.js**, pero si queremos utilizarlo, por ejemplo, en **main.js** tendremos que **exportarlo** de su propio archivo.

Para realizar esta exportación le añadiremos la palabra reservada **export** a dicha constante:

```
export const person = {  
  name: "Stephen",  
  surname: "Strange"  
};
```

De esta manera, esta constante ya podrá “salir” de su archivo `modules.js`.

Ahora que tenemos lista nuestra constante, vamos a volver a el fichero **main.js** e importar dicha constante con destructuring (ya que gracias a estos métodos podemos manejar nuestros ficheros como si fueran objetos):

```
import './style.css'
import javascriptLogo from './javascript.svg'
import { setupCounter } from './counter.js'
import { person } from './modules';
[...]
```

Si tras este paso, y en el propio archivo **main.js** hacemos un `console.log` de la constante `person` importada del archivo **modules.js** podremos ver que efectivamente nos trae la información de nuestro objeto, por lo que en este punto tenemos de manera aislada una parte de un script que podemos utilizar al exportarlo en cualquier fichero JavaScript que tengamos en nuestra aplicación.

Vamos a probar con otro ejemplo con una función:

```
export const person = {
  name: "Stephen",
  surname: "Strange",
};

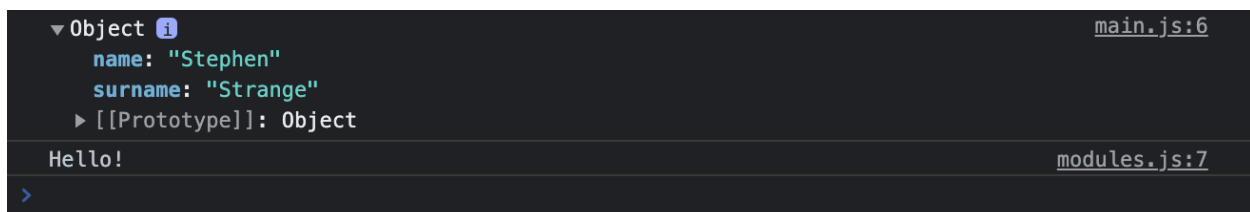
export const sayHello = () => {
  console.log("Hello!");
};
```

En este punto estamos exportando de nuestro fichero **modules.js** el objeto `person` y la función `sayHello`. Vamos a ver si lo podemos importar de la misma manera y ejecutar dicha función:

```
import './style.css'
import javascriptLogo from './javascript.svg'
import { setupCounter } from './counter.js'
import { person, sayHello } from './modules';

console.log(person);
sayHello();
```

Si inspeccionamos la consola del navegador podremos comprobar que efectivamente todo está funcionando a la perfección:



Hay otro método para exportar e importar información sin destructuring llamado **export default**, esto lo que hace es exportar por defecto una constante.

Esto lo que hará es, por un lado, permitirnos importar sin destructuring la información que exportemos de esta forma y, por otro lado, habilitar el renombrado de esta información en el fichero donde lo importemos.

```
export const person = {
  name: "Stephen",
  surname: "Strange",
};

export const sayHello = () => {
  console.log("Hello!");
};

const DEV_LAN = "JavaScript";

export default DEV_LAN;
```

En nuestro **module.js** vamos a exportar por defecto la constante **DEV_LAN** , por lo que vamos a importar esta misma constante en **main.js**:

```
import './style.css'
import javascriptLogo from './javascript.svg'
import { setupCounter } from './counter.js'
import DEV_LAN, {person, sayHello} from './modules';

console.log(person);
sayHello();
console.log(DEV_LAN)
[...]
```

Como véis, la constante exportada por defecto la podremos importar sin hacer destructuring, separando a la hora de importar las constantes exportadas y las exportadas por defecto.

Tal y como hemos mencionado anteriormente, podemos renombrar las constantes exportadas por defecto de la siguiente forma.

```
import './style.css'
import javascriptLogo from './javascript.svg'
import { setupCounter } from './counter.js'
import mainLanguage, {person, sayHello} from './modules';

console.log(person);
sayHello();
console.log(mainLanguage)
[...]
```

Si comprobamos el `console.log` de **mainLanguage** veremos que se trata de la misma constante que hemos exportado por defecto, pero que hemos renombrado a la hora de importarla en el fichero **main.js**.

Por convención, la información exportada por defecto es la información principal de nuestro fichero, siendo las exportadas sin más las "opcionales" o "utilidades" adicionales del mismo fichero.

Para empezar a trabajar con este sistema de archivos recomendamos utilizar el export normal, ya que si es el renombrado una de las características más llamativas del export default, el export normal también nos lo permite de la siguiente forma.

```
import DEV_LAN, {person as personObject, sayHello} from './modules';

console.log(personObject);
```

Como podeis comprobar, gracias a la palabra reservada **as** le estamos asignando un **alias** a nuestra constante importada, renombrando de una manera más “suave” dicha constante y pudiendo utilizar su alias en nuestro fichero.

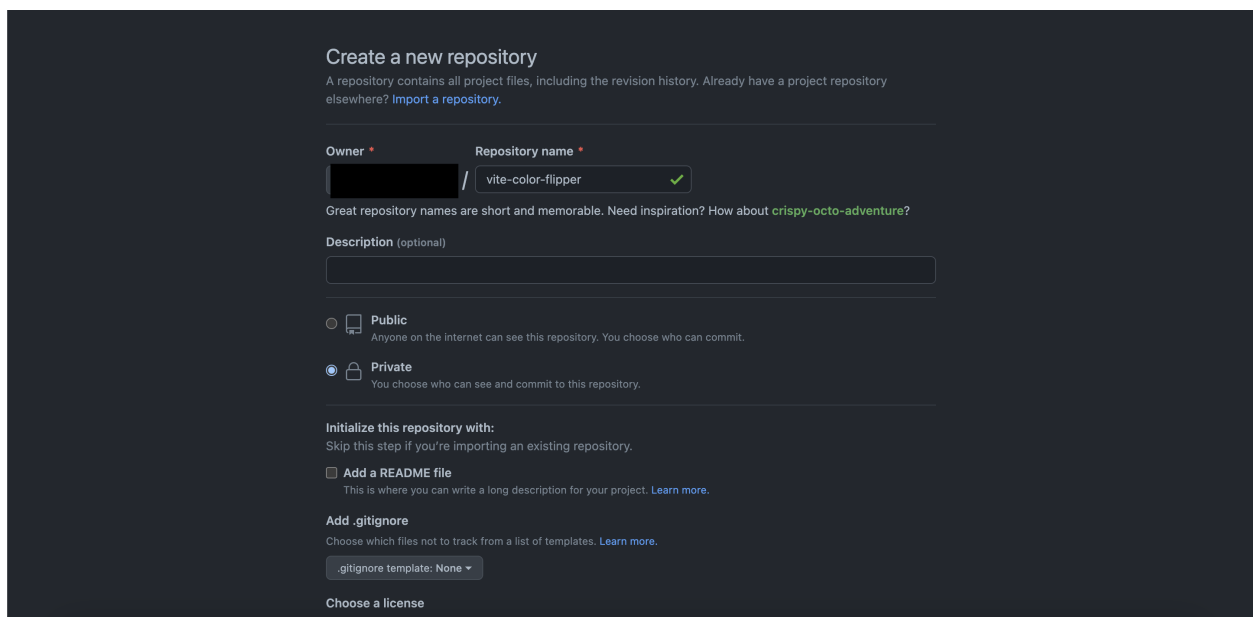
Importante, estamos renombrando como `personObject` a `person` en el archivo `main.js`, pero no estamos sustituyendo su nombre original. Aunque no tenga mucho sentido, podemos ponerle un apodo diferente en cualquier fichero donde importemos esta constante.

Además de esto, tenemos que tener en cuenta que si renombramos una constante a través del `import` no podremos usar su nombre original en ese fichero, porque estamos importandonos, en el caso del ejemplo, `person` COMO `personObject`.

Deploy

Como último paso vamos a desplegar nuestra aplicación web en un dominio y así poder visitarla desde cualquier dispositivo a través de cualquier navegador.

Lo primero que tenemos que tener es un repositorio de **GitHub** con nuestro proyecto subido.

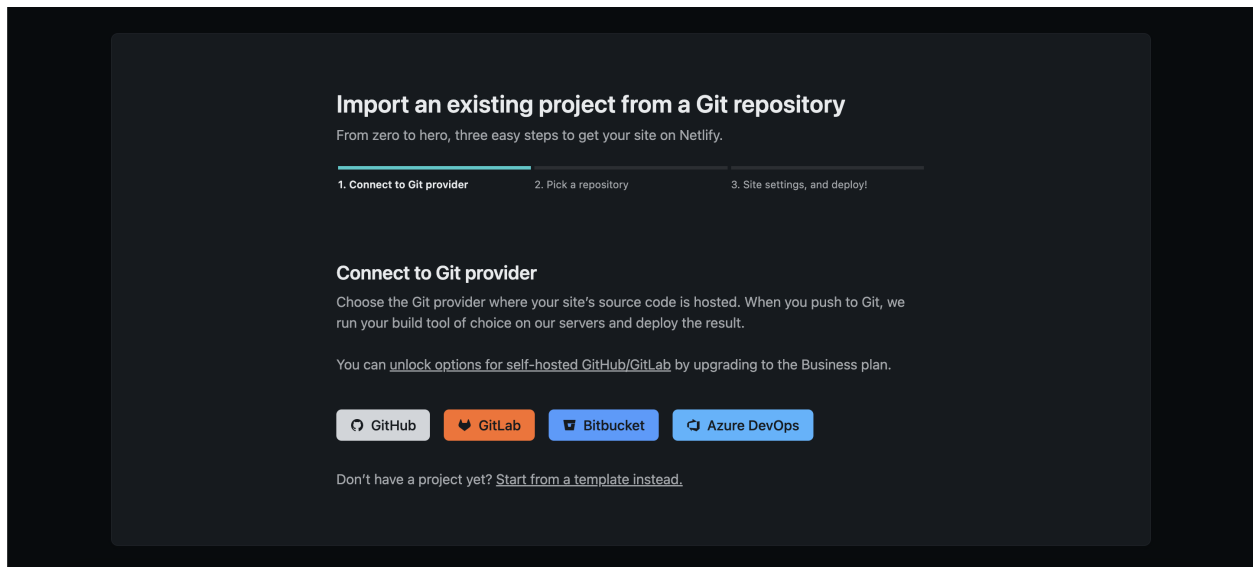
The image shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two input fields: 'Owner' (which is partially obscured by a black box) and 'Repository name' (which contains 'vite-color-flipper' and has a green checkmark next to it). A hint suggests repository names should be short and memorable, with an example 'crispy-octo-adventure?'. There is a 'Description (optional)' text area below. The 'Visibility' section has two radio buttons: 'Public' (selected) and 'Private'. The 'Initialize this repository with:' section has a checkbox for 'Add a README file' (which is checked) and a dropdown for '.gitignore template' set to 'None'. At the bottom, there is a 'Choose a license' section.

El objetivo es subir a un nuevo repositorio todos los archivos a través de `git` para tener el proyecto tal y como está en la carpeta raíz creada por Vite. Para ello haremos un `commit` de con todos los ficheros y carpetas (`gitignore` se encargará de no subir la carpeta `node_modules`) y hacer un `push` a la rama principal.

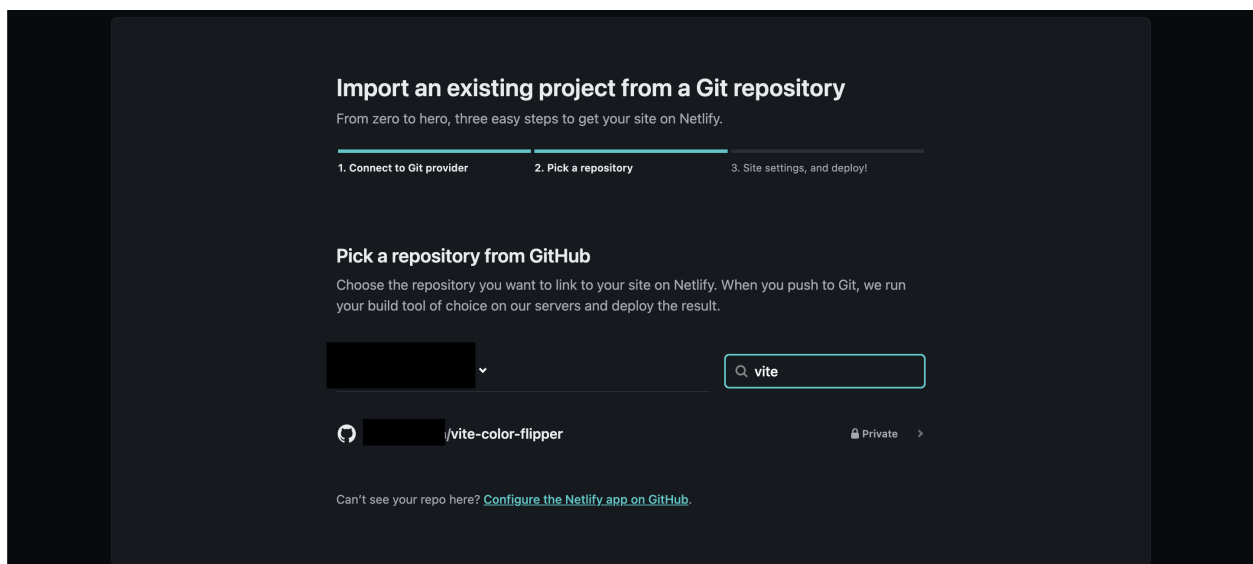
Una vez hecho esto, la web que utilizaremos para desplegar nuestro proyecto en este caso será **Netlify**.

Vamos a loguearnos con **GitHub** y autorizar a Netlify para que pueda acceder a todos nuestros repositorios, de esta forma podremos elegir qué queremos desplegar a través de la plataforma.

Haremos click en **New Site** o **Import an existing project** y pasaremos a una serie de pantallas que nos irán guiando durante el proceso de despliegue:



En este punto haremos click en GitHub y le damos la autorización mencionada.



Buscamos nuestro repositorio con el proyecto que hemos subido anteriormente.

Branch to deploy

master

Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

[Learn more in the docs](#)

Base directory

Build command

npm run build

Publish directory

dist

Show advanced

Deploy site

Por último tendremos que indicarle las instrucciones del despliegue.

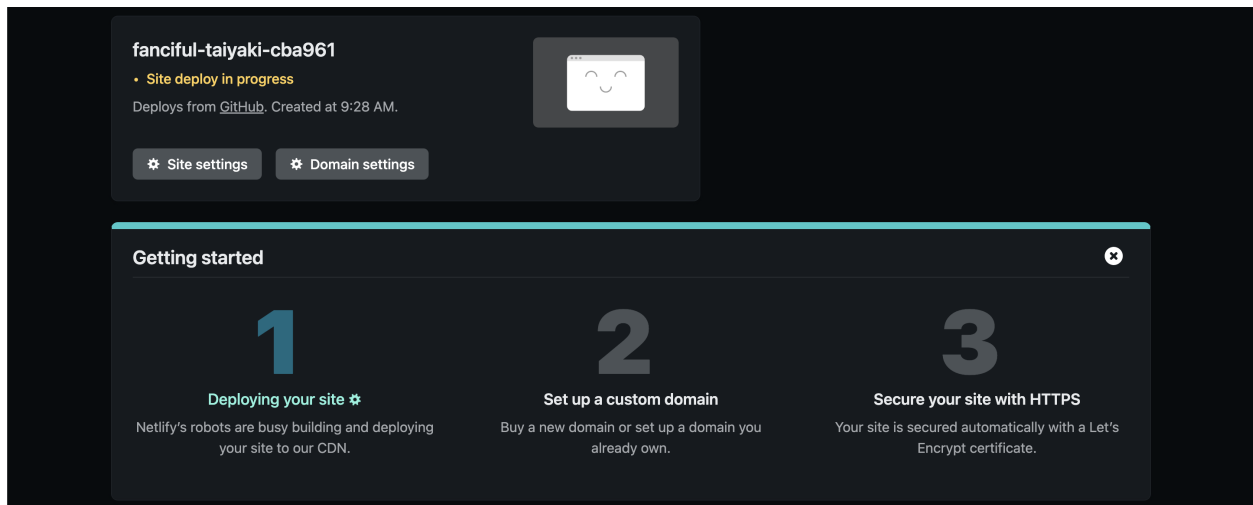
El primer parámetro será la **rama** a desplegar, que en nuestro caso es la rama master del repositorio.

El segundo parametro es el **directorio base** donde se aloja la raíz de nuestro proyecto, en el repositorio los archivos principales están en la propia raíz y no anidado en alguna carpeta extra, por lo que no tenemos que indicar nada en ese campo.

El tercer parámetro es el comando para realizar el **build** del proyecto. Como vimos anteriormente, Vite ya incluye en el package.json un script que realizará la build a través de **npm run build**, por lo que no hace falta que modifiquemos lo que nos indica por defecto.

Y por último el directorio donde se encontrará la build, que de igual forma vimos que genera una carpeta llamada **dist**, por lo que tampoco necesitamos modificar el valor de este campo.

Ya podemos clicar en **Deploy Site** y veremos el proceso del despliegue, el cual nos informará si hay algún error o inconveniente durante el proceso.



Si todo ha ido correctamente, Netlify nos habrá generado una URL con el proyecto desplegado que podremos compartir con quién queramos y visitar nuestro proyecto desde cualquier dispositivo 🥳.