

# Divide y vencerás

*“Escribe código que sea fácil de eliminar, no fácil de extender”*

Tef,— Programming is Terrible

Durante la primera mitad de este sprint nos hemos dedicado a estudiar y practicar Javascript. Entender este lenguaje de programación es un paso clave hacia la construcción de aplicaciones web debido a que él será el que se encargue de coordinar las interacciones y acciones que ocurrirán en la puesta en marcha de una aplicación.

En esta toolbox te contaremos y profundizaremos en cómo utilizar **módulos** para separar e independizar las piezas que conforman una aplicación.

## La utopía del desarrollo

El código fuente ideal de una aplicación tiene una estructura clara y transparente. Funciona de forma concisa, su lectura es digerible, y el código es fácil de explicar; cada parte tiene un rol perfectamente definido.



Un programa típico crece de forma orgánica, se añaden nuevas piezas o funcionalidades según aparezcan nuevos requerimientos. Piensa en esto: ¿cómo construirías tu casa si tuvieras un terreno grande? Después de todo, [hacer software es como hacer una casa](#).

Tener una estructura (y preservarla) requiere un trabajo adicional, el cual se verá reflejado en el futuro cuando otra persona tenga que agregar algo a esa



estructura creada. Puede ser tentador el ser negligentes con este aspecto y codear para que “funcione en el momento”, pero es a riesgo de hacer un gran enredo para futuros programadorxs. Ten mucho cuidado con esto, por que esa persona futura puedes ser tú mismo. Es muy común que el código que haces hoy no lo reconozcas mañana, así que siempre codea de forma que cualquier persona pueda entenderlo más adelante.

Hasta ahora hemos venido desarrollando aplicaciones en un mismo archivo, lo cual refleja la complejidad que hemos venido manejando, pero este escenario no podrá, y no deberá, ser replicado en aplicaciones grandes compuestas de muchas partes y muchas piezas.

## Contextos

El trabajar sobre un mismo archivo tiene sus ventajas, sobre todo en el proceso de aprendizaje en el que nos encontramos. Al tener todo nuestro código en un archivo tenemos control y visualización sobre los valores y cómo estos se relacionan entre sí. Podemos ver qué variables están siendo utilizadas por funciones, y cómo están siendo modificadas.

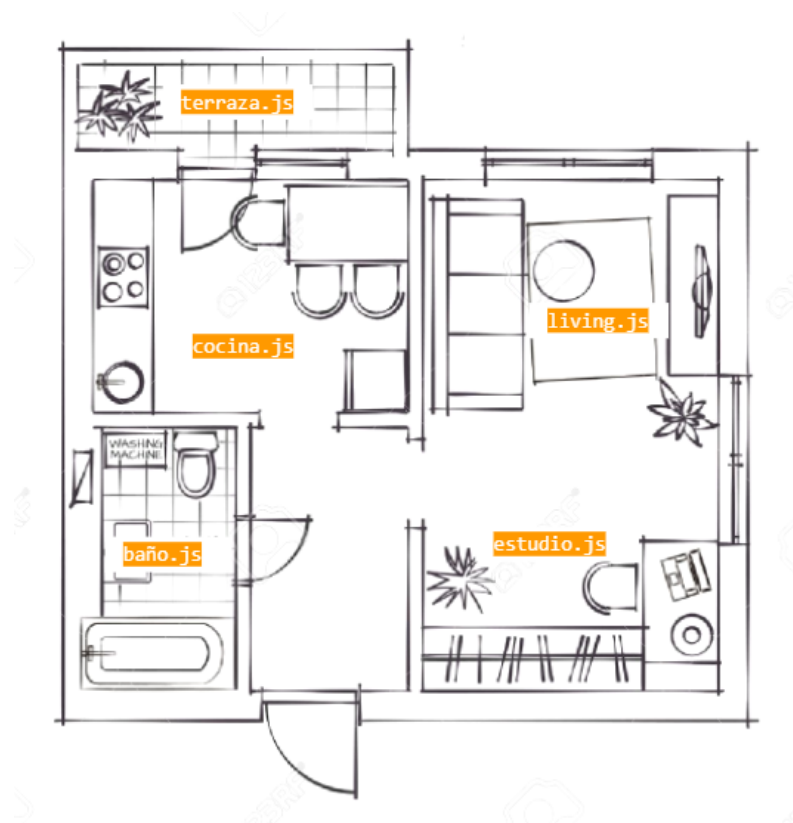
Pero como te mencionamos, este escenario es atípico en proyectos grandes. Una aplicación sencilla puede llegar a tener más de 1 archivo, haciendo que esta idea de control y visualización se difumine un poco, especialmente por que los distintos archivos a trabajar no serán desarrollados por la misma persona.

Cuando desarrollamos con Javascript, todo lo que hagamos por defecto comparte un mismo contexto. El contexto **no es más que las relaciones de disponibilidad entre valores**. ¿Te parece muy técnica esta explicación? no te preocupes. Para entender mejor este concepto vamos a poner en práctica nuestra imaginación: piensa que estás en una reunión en tu casa con 3 de tus mejores amigxs.





Además, imagina tu casa como un conjunto de archivos de Javascript (si, muy matrix esto):



Adicionalmente, visualizate a ti y a tus amigxs como objetos de Javascript:



Ahora bien, estando en esta situación, en nuestra casa con nuestrxs amigxs, es correcto afirmar que cualquier actividad grupal que vayamos a realizar (jugar a la switch, hacer unas hamburguesas, ver un partido de fútbol) la podremos hacer llamando a las personas necesarias al lugar de ejecución, es decir, que todas las personas involucradas puedan escucharnos si gritamos su nombre en voz alta, ¿cierto?. Cuando decimos que el contexto **no es más que las relaciones de disponibilidad entre valores**, nos referimos precisamente al hecho de que distintas variables estarán siempre disponibles bajo un escenario determinado.

En esta metáfora la aplicación web es tu casa, cada cuarto representa un archivo, y cada persona es un valor. Cualquier persona puede acceder a cualquier lugar de tu casa siempre y cuando se lo llame ahí. Ahora, veamos esta situación en código:

### [Contexto en reunión](#)

## Tipos de contexto

Como te comentamos previamente, el **contexto** puede definirse como la disponibilidad que tendrá una variable dentro de la aplicación.

Existen dos tipos de contextos: local o global.

- Las variables con **contexto local** son las que se crean dentro de una función y que sólo pueden ser accedidas desde su propia función o funciones anidadas.
- Las variables con **contexto global** son las que se declaran fuera de las funciones y pueden ser accedidas desde cualquier parte del código.



Ya te hemos contado sobre dos declaraciones de variables en Javascript: ``let`` y ``const``. También existe una tercera forma llamada ``var``, que fue la primera, y única, forma que tenía Javascript de declarar variables. Ten en cuenta que actualmente está en desuso.

Las variables ``var`` son automáticamente variables globales, sin importar donde se declaren, ya sea tanto fuera o dentro de una función. Esto se considera una mala práctica. Sin embargo, esta opinión varía entre la comunidad. En [este artículo](#) un programador nos cuenta por qué dejó de usar ``var`` en sus proyectos.

No obstante, el problema de las variables globales no se resuelve simplemente dejando de usar ``var``. En el video anterior hicimos una demostración de cómo obtenemos un problema similar con ``let``, lo cual nos indica que la globalidad de una variable no depende solo del lenguaje utilizado, sino también del diseño en general de una solución. En conclusión, podemos decir que [hay que evitar a toda costa](#) las variables globales. En el siguiente video ampliamos un poco más sobre los distintos tipos de contexto:

### [Tipos de contexto](#)

Ante estas situaciones, donde las aplicaciones crecen a un punto de necesitar varios archivos, los **módulos de Javascript** nos ayudarán a independizar piezas y partes del código en espacios individuales, los cuales luego podremos conectar para construir funcionalidades de una aplicación.

## Módulos

Entonces, para solventar el problema del contexto global, llegan los módulos de Javascript, los cuales nos van a permitir aislar por completo una o un conjunto de funcionalidades y variables dentro de un mismo archivo.

Un módulo es una parte del programa que se encuentra en aislamiento y lista para ser usada. También define sus relaciones respecto de otros módulos. Cuando un módulo necesita una pieza de otro módulo, decimos que hay una **dependencia**.





Crear módulos en Javascript es bastante simple, ya que un módulo nuevo equivale a un archivo nuevo. La consideración que debemos tener al trabajar con módulos es especificar la relación entre ellos, y eso lo vamos a hacer con dos palabras reservadas: ``import`` y ``export``.

Para hacer disponible un módulo para nuestra aplicación, tendremos que **exportarlo**, y si quisiéramos utilizar este módulo dentro de otro, tendríamos que **importarlo**.

Existen varias formas de importar y exportar archivos, pero por ahora nos vamos a concentrar en los export nombrados (*named export*) y los export por defecto (*default export*).

Los *named exports*, o exports nombrados, los utilizamos cuando tenemos varias cosas que queremos exportar dentro de un mismo archivo. El siguiente video nos demuestra cómo usar esta característica de Javascript.

[Named exports](#)

El *default export*, o export por defecto, se usa cuando tenemos un módulo que consta de una función y queremos exportar sólo esa función. Observa el siguiente video para ver una demostración de esto.

[Export default](#)



## Resumiendo


Los módulos de Javascript nos dan una forma de independizar nuestro código en sus propios contextos. Esto nos trae como beneficio la separación de las funcionalidades desarrolladas y la habilidad de poder conectarlas en los lugares en los que sean necesarios.


Modularizar una aplicación es un paso esencial en la extensión del software. Al aislar las funcionalidades en sus propios contextos podemos reutilizarlas en distintos lugares de nuestras aplicaciones, garantizando y anticipando su uso correcto.

[Nos tomamos unos minutos para completar esta encuesta.](#) Queremos saber cómo valoran mi tarea hasta acá. ¡Les va a llevar solo un minuto!


## ¡Prepárate para la próxima meeting!

### Profundiza


 Los módulos de Javascript han pasado por un proceso de maduración para llegar a donde están hoy. [Este artículo](#) nos cuenta una breve historia sobre el uso de los módulos en este lenguaje.

 [Este tutorial de Digital Ocean](#) nos muestra casos de usos similares para los named y default exports.

### Challenge

 ¿Recuerdas el challenge de la toolbox anterior? Partiendo de [este sandbox](#), sigue las instrucciones en cada archivo para desarrollar la aplicación. Deberás poner en práctica lo aprendido en esta toolbox para modularizar cada funcionalidad.

### Comunidad

 FreeCodeCamp es popular por su gran forum de preguntas y respuestas, donde diversas personas de la comunidad intercambian información sobre



temas de programación. Además, ofrece una gran lista de recursos para aprender sobre estos mismos temas. Explora el concepto de módulos de Javascript en [este artículo](#).

