



Precarga de módulos 16/25



RECURSOS

APUNTES

Modularizar una aplicación nos beneficia en rendimiento gracias a las técnicas de *Lazy Loading* y *CodeSplitting* pero... ¿Funciona realmente? ¿El rendimiento de mi app será el óptimo?

Diferencias entre hacer CodeSplitting y sin CodeSplitting

Al realizar un `ng serve`, SIN *CodeSplitting*, puedes observar en la consola algo como lo siguiente:

Initial Chunk Files	Names	Size
vendor.js	vendor	2.36 MB
polyfills.js	polyfills	123.40 kB
main.js	main	39.79 kB
runtime.js	runtime	6.63 kB
styles.css	styles	1.18 kB
Initial Total		2.52 MB

Mientras que al realizarlo luego de haber modularizado la app se observa lo siguiente:

Initial Chunk Files	Names	Size
runtime.js	runtime	12.56 kB
main.js	main	11.46 kB

Lazy Chunk Files	Names	Size
src_app_modules_website_website_module_ts.js	-	20.20 kB
src_app_modules_cms_cms_module_ts.js	-	7.89 kB

En la primera imagen, toda tu aplicación se agrupa en el `main.js` y en la segunda, luego de modularizar, se crea un “*Chunk*” por cada módulo que tenga la app.

Hasta aquí ya puedes observar la diferencia en el tamaño de los archivos. Si bien estamos compilando la aplicación en modo desarrollo. En modo producción se comprime aún más y pesarán menos los archivos.

Diferencias entre hacer Lazy Loading y sin Lazy Loading

Al ingresar a tu aplicación desde el navegador, utiliza las herramientas de desarrollo que proveen para inspeccionar la red.

Sin *Lazy Loading*, solo se puede observar el archivo `main.js` que contiene toda tu aplicación.

Status	Method	Domain	File
200	GET	localhost:4200	runtime.js
200	GET	localhost:4200	polyfills.js
200	GET	localhost:4200	vendor.js
200	GET	localhost:4200	main.js

Con *Lazy Loading*, puedes observar que se desprende un nuevo archivo llamado `src_app_modules_website_website_module_ts.js` el cual contiene el módulo Website.

Status	Method	Domain	File
200	GET	localhost:4200	runtime.js
200	GET	localhost:4200	polyfills.js
200	GET	localhost:4200	vendor.js
200	GET	localhost:4200	main.js
200	GET	localhost:4200	src_app_modules_website_website_module_ts.js

Te preguntarás, ¿dónde están los archivos correspondientes a mis otros módulos?

Redirrecciónate a una ruta de tus otros módulos para observar como se carga dinámicamente el archivo Javascript.

Status	Method	Domain	File
200	GET	localhost:4200	runtime.js
200	GET	localhost:4200	polyfills.js
200	GET	localhost:4200	vendor.js
200	GET	localhost:4200	main.js
200	GET	localhost:4200	src_app_modules_website_website_module_ts.js
200	GET	localhost:4200	src_app_modules_cms_cms_module_ts.js

Esa es la magia del *Lazy Loading*. Has logrado que tu aplicación cargue en N cantidad de partes, reduciendo el peso de la misma y mejorando su velocidad de renderizado.

Y seguro has entendido mejor lo que es *Lazy Loading*, *CodeSplitting* y cómo trabajan juntos.

Contribución creada por: Kevin Fiorentino.

Archivos de la clase

