

Curso de Angular Router: Lazy Loading y Programación Modular



Precarga de módulos 16/25

 $rac{1}{2}$ 

**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 vendor.js polyfills.js main.js runtime.js styles.css	Names vendor polyfills main runtime styles	Size 2.36 MB 123.40 kB 39.79 kB 6.63 kB 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
                                                              Size
                                                Names
                                                runtime
                                                          12.56 kB
runtime.js
main.js
                                                          11.46 kB
                                                main
Lazy Chunk Files
                                                              Size
                                                Names
src app modules website website module ts.js
                                                          20.20 kB
src app modules cms cms module ts.js
                                                           7.89 kB
```

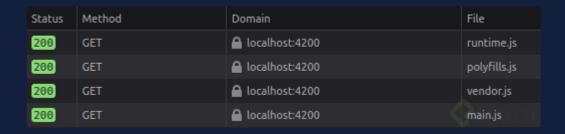
En la primera imagen, toda tu aplicación se agrupa en el main.js y en la segunda, luego de modularizar, se crea un "Chunk" por cada módulo que tenga la app.

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.



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	<b>△</b> localhost:4200	src_app_modules_website_website_module_ts.js

Te preguntarás, ¿dónde están los archivos correspondientes a mis otros módulos? Redirecciónate a una ruta de tus otros módulos para observar como se carga dinamicamente el archivo Javascript.

Status	Method	Domain	File
200	GET	<b>△</b> 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



