Velocidad y SEO con el SSR de Angular Universal

Las *SPA JavaScript*, muy balanceadas hacia el navegador, nacieron para **crear con tecnología web aplicaciones de negocio**. Normalmente se desplegaban en *intranets*, o en internet para usuarios autorizados. Eran aplicaciones de uso intensivo, visita recurrente y alto rendimiento diario. El éxito tecnológico de *frameworks* como Angular las llevó a ser usadas para desarrollar webs clásicas de internet y ser utilizadas por visitantes ocasionales.

Pero en esta situación presentaron dos problemas para los que inicialmente no estaban preparadas. Por un lado la primera visita de un humano obligaba a la descarga completa de la aplicación antes de poder ver nada. Y nada era lo que veían los visitantes robóticos que pretendían indexar un sitio. Las soluciones a estos problemas incluyen, entre otras medidas, **una vuelta al servidor**. Lo que en Angular se conoce como **aplicación universal**.

Partiendo del código tal como quedó en PWA, Entre la web y las apps con Angular. Al finalizar tendrás una aplicación que se instala, actualiza y comporta como una aplicación nativa.

> Código asociado a este tutorial en *GitHub*: AcademiaBinaria/angular-boss

1 Angular Universal

1.1 Vuelta al servidor

> Tenemos: Angular nació para vivir en el navegador.

- Para **quitarle carga al servidor** generando el contenido dinámico en el navegador en base a plantillas.
- Para enviar por la red primero la aplicación y después los datos, **ahorrando transferencia** durante un uso continuado.
- Para mejorar la **experiencia del usuario** al no percibir recarga de página durante la navegación dentro de la aplicación.

Ideal en entornos de intranet o aplicaciones de gestión de uso intensivo.

Problemático para uso esporádico o indexable públicamente.

> Queremos: Indexado SEO y velocidad en la primera visita y también en las sucesivas.

1.2 Para mejorar el SEO

> El contenido se genera durante la ejecución del JavaScript en el navegador.

- Los **robots** no tienen nada significativo que indexar.
- Las **redes sociales** no encuentran cabeceras para mejorar la presentación de enlaces.

Hay que enviar el contenido ya generado.

Pero sin perder la experiencia de usuario durante la ejecución.

1.3 Para mejorar la experiencia en la primera visita

> Para mostrar contenido antes hay que descargar y ejecutar la aplicación.

- Los **usuarios** ven una página vacía demasiado tiempo.
- El **peso de la descarga inicial** es desproporcionado a pesar de *lazy loading*.

Hay que enviar el contenido ya generado.

Descargar la aplicación en segundo plano.

2 Despliegue con Node Express

El reto está en mantener lo bueno de las aplicaciones *JavaScript* como es la transición fluida entre páginas, la interactividad o la descarga de datos bajo demanda, pero combinado con una mejor primera experiencia. Para ello la descarga del `index.html` tiene que venir con un documento html ya preparado con algo para mostrar y tardar lo menos posible en permitir la interacción.

El **tiempo para el primer pintado** se ve penalizado por el tamaño del *bundle* principal de Angular, pues en él reside habitualmente el componente *app* que actúa de raíz. Por supuesto que utilizar la carga diferida de módulos es una manera obligada de reducir el peso del *main*. Todas las rutas, incluida la ruta base, deben ser *lazy* para retrasar la navegación y que la descarga se produzca más tarde. Sólo el componente raíz con la *shell* de navegación básica debería venir en el *bundle* principal.

Pero ni con eso es suficiente. El usuario no verá nada hasta que Angular se descargue, reclame el *bundle main*, lo procese y renderice ese *shell*. Hay usar alguna estrategia extra para reducir el tiempo de espera y entretener al usuario.

2.1 Add Express Engine

Claro que esto es sólo un truco para que ese primer momento de espera se reduzca y no perdamos potenciales visitantes. Si queremos algo más, como por ejemplo que el contenido a descargar sea más fresco, entonces necesitaremos **renderizar en el servidor**. Y para ello necesitaremos un servidor de verdad. El propuesto y mejor documentado es *Express de NodeJS*.

Para empezar tendrás que instalar y registrar las librarías necesarias. Además habrá que crear el pequeño servidor *Express*, y configurar al CLI para que haga el despliegue de ambos: cliente y servidor. Este laborioso trabajo se ha automatizado y ahora mismo se resuelve casi todo con una instrucción.

```
ng add @nguniversal/express-engine --clientProject shop
```

2.2 Scripts de compilado y despliegue

Cuando termina la generación e instalación, comienza el compilado. En este caso es doble, porque además de la compilación habitual que llamaremos *client-bundle*, habrá que compilar el servidor con su propia versión de

la aplicación cliente. Será el *server-bundle*.

Además necesitamos una tercera compilación (aunque no es propiamente de Angular) para el servidor *node/express* que ejecutará el *server-bundle*. Y por último ya sólo nos queda poner en marcha dicho servidor que se quedará a la espera de peticiones de páginas.

Todo esto se resume en los siguientes scripts, casi todos auto generados por el CLI.

```
{
  "start:ssr": "npm run build:ssr && npm run serve:ssr",
  "build:ssr": "npm run build:client-and-server-bundles && npm run
compile:server",
  "build:client-and-server-bundles": "ng build --prod && ng run
shop:server:production --bundleDependencies all",
  "compile:server": "webpack --config webpack.server.config.js --progress --
colors",
  "serve:ssr": "node dist/server"
}
```

2.3 Control de rutas

El resultado es un servidor *Node* que a cada petición web responde enviando el `index.html`. Pero, y esta es la clave, resolverá la ruta ejecutando la aplicación Angular antes de responder al navegador. De esa forma el `index.html` irá recién generado con el contenido tal cual lo vería el usuario tras la ejecución de Angular en local. Así que **la espera al primer pintado significativo se reduce** y eso es bueno.

Por si fuera poco, la principal ventaja al usar este método es que al traer información dinámica puede usarse para indexar el contenido real del sitio. Esto es doblemente bueno, porque ahora todos **los robots indexadores podrán catalogar tu *site*** como si de una web clásica se tratase. Y los usuarios humanos podrán continuar la ejecución en local disfrutando de las ventajas de una SPA.

El trabajo del servidor a partir de ese momento será mayúsculo. Recibe la petición, invoca a su versión de nuestra aplicación Angular y la ejecuta en memoria. El resultado es un documento HTML que devolverá al usuario. Dará lo mismo que ruta se le pida y si se resuelve por lazy-loading. Va totalmente preparado y configurado para cargar el JavaScript oportuno y ejecutarlo como lo haría un navegador.

Compruébalo solicitando diversas rutas e inspeccionando la respuesta del servidor.

```
http://localhost:4000
http://localhost:4000/rates
```

De todas formas tengo que advertirte de que tomes todo esto con cautela por varios motivos:

- Tecnología compleja estable pero con carencias
- Herramientas de generación buenas pero incompletas
- Transferencia de estado manual para evitar llamadas repetidas al API

Tampoco es sencilla la convivencia con librerías propias del *browser*, y menos si se trata de una PWA. Yo procuro usar un servicio que aisle al servidor de ciertas llamadas sólo disponibles en el navegador, como por ejemplo todo lo relativo al *localStorage*.

```
import { isPlatformBrowser, isPlatformServer } from '@angular/common';
import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class UniversalService {
  constructor(@Inject(PLATFORM_ID) private platformId: string) {}
  public isBrowser() => isPlatformBrowser(this.platformId);
  public isServer() => isPlatformServer(this.platformId);

  public saveOnStorage(key, value) {
    if (this.isBrowser()) {
      sessionStorage.setItem(key, value);
    } else {
    }
  }
  public loadFromStorage(key) {
    if (this.isBrowser()) {
      sessionStorage.getItem(key);
    } else {
      return null;
    }
  }
}
```

3 Variantes: shell y pre-rendering

La técnica vista anteriormente resuelve los problemas de usuario y de robot SEO, pero a costa de cierta complejidad. Y, sobre todo, a costa de necesitar un servidor node en producción.

En algunas situaciones queremos desplegar la aplicación en un sencillo servidor de ficheros. A veces ni siquiera tenemos la necesidad del indexado completo.

Para esos caso tenemos alternativas más sencillas.

3.1 Shell para mejora de experiencia inicial

- Muestra un contenido instantáneo mientras descarga la app.
- Mejora la experiencia de usuario en la primera visita.
- De cara al SEO, sólo indexa el contenido inicial.

> Adecuado para aplicaciones de usuario registrado, pero con un portal de bienvenida indexable y rápido.

La más sencilla es hacer que el `index.html`, habitualmente vacío, se rellene con un **contenido visualizable mientras el proceso principal de Angular no arranca**. En ocasiones basta con poner a mano algún mensaje o animación que indique que estamos cargando. Pero cuanto más se parezca esa primera visión al resultado final mejor para el usuario. Así que lo propio sería que el `index.html` ya bajase con el *shell* real de la aplicación.

Montar eso a mano no es la mejor opción. La solución parte de **renderizar el *html* durante el proceso de *deploy***. Se trata de configurar el CLI para que ejecute la aplicación en la máquina del desarrollador sobre una ruta predefinida; y que copie el resultado sobre el `index.html` que enviará a distribución.

Este trabajo se ha automatizado y se resuelve con un par de comandos del Angular CLI.

```
ng g app-shell --client-project shop --universal-project server-shop
```

El efecto de este comando se materializa especialmente con la aparición de nuevos *targets para los builders* del CLI en el fichero `angular.json`.

Con el comando `ng run shop:app-shell` podrás generar una versión especial de distribución en al que el `index.html` ya va prerenderizado con el contenido del componente asociado a la ruta *shell*. Realmente lo que hace es ejecutar tu aplicación sobre una ruta predefinida, tomar el *html* resultado e inyectarlo en el *body* del `index.html` que irá a distribución.

> Por cierto, esta técnica no obliga a disponer de ningún servidor web especial. Sigue funcionando con un servidor estático de ficheros pues la prerenderización se produjo en la máquina del programador.

3.2 Pre renderizado de toda la aplicación

- Se trata de volver no sólo al servidor web, si no al servidor de ficheros.
- La idea es invocar repetidamente al SSR y almacenar el HTML resultante en ficheros físicos.
- En producción, los robots y los usuarios recibirán ya esas copias pre generadas.
- Para mantener el sistema actualizado se necesita regenerar frecuentemente los ficheros

> Adecuado para blogs y sitios que no puedan o no quieran tener un servidor web corriendo.

A falta de una solución oficial, podemos seguir la pista a iniciativas tipo Angular Prerender

4 SEO en la página, en el navegador y en el servidor

Con lo visto hasta ahora tu aplicación estará más que cubierta en cuanto a ofrecer la mejor experiencia para usuarios al tiempo que envía contenido indexable para robots... Pero falta algo.

Habitualmente las aplicaciones Angular manejan el contenido visible de una página web; es decir, el *body*. Para **acceder y cambiar el contenido del *header***, tan utilizado por los robots de redes sociales, hay que usar algo más.

4.1 Titulo y meta etiquetas de página

Como parte del *framework* viene la librería *platform-browser* dónde tenemos un par de servicios para manipular el título y cualquier etiqueta de meta información de la página.

Para ello suele usarse un código similar a este en el componente raíz de la aplicación. Es muy sencillo pero te dará una idea del potencial de estos servicios:

```
import { Meta, Title } from '@angular/platform-browser';
@Component({
  selector: 'app-root',
  template: `<p>Aprende a usar el framework Angular</p>`,
})
export class AppComponent implements OnInit {
  constructor(private title: Title, private meta: Meta) {}
  ngOnInit() {
    this.title.setTitle('My title');
    this.meta.addTag({ property: 'og:title', content: 'My title' }, true);
  }
}
```

Ahora ya tienes una aplicación que satisface a usuarios y robots por igual. Continúa tu formación avanzada para crear aplicaciones Angular.

> Aprender, programar, disfrutar, repetir. -- *Saludos, Alberto Basalo*