PWA, Entre la web y las apps con Angular

Las aplicaciones web han evolucionado desde el contenido estático al dinámico, luego al adaptable y ahora se acercan **progresivamente a la experiencia y rendimiento de aplicaciones nativas**. En Angular se incorporan esas capacidades desde el propio *framework*.

Mediante la librería `@ngular/pwa` dispondremos de todo lo necesario para ofrecer instalación en escritorio, actualizar la aplicación controladamente, recibir notificaciones del servidor e incluso mejoras en la caché de comunicaciones que pudieran permitir un funcionamiento *offline*.

Partiendo del código tal cómo quedó en El patrón Redux con NgRx en 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 Las herramientas de la librería PWA

Aprovechando las nuevas características del *Angular CLI* tenemos el comando `ng add @angular/pwa` que instala los paquetes y configura cualquier aplicación Angular convirtiéndola en una *PWA* básica.

Estos son los actores y ficheros involucrados en este proceso tras la agregación del paquete `pwa`.

- ServiceWorkerModule
- ./angular.json { "serviceWorker": true }
- ./ngsw-config.json
- ./src/manifest.webmanifest
- ./src/index.html
- ./assets/icons/..

Con todo instalado y configurado, querrás probar que funciona. Pero, a primera vista no ocurre nada. **Las características *PWA* están preparadas para funcionar en modo producción**. Así que lo primero será ejecutar el comando `ng build --prod` y lanzar un servidor sobre la carpeta de distribución recién creada.

De entre los nuevos ficheros que aparecen hay dos que debes conocer especialmente, **el *manifest* y el *service worker***.

1.1 El manifest.webmanifest

Este sencillo documento acompaña al `index.html` y le da indicaciones al navegador para que trate esta aplicación de manera especial. El fichero en sí contiene una *metadata* con textos descriptivos, colores e iconos para ser usados por el navegador y el sistema operativo y **ofrecerle al usuario que instale la *web* como una *app***.

Casi todo es cosmético, pero merece la pena prestar especial atención a los iconos. Por defecto el *CLI* instala distintas versiones del logo oficial de Angular. Una forma cómoda de sustituirlo por los tuyos es usar la herramienta *ngx-pwa-icons*

```
{
  "name": "shop",
  "short_name": "shop",
  "theme_color": "#1976d2",
  "background_color": "#fafafa",
  "display": "standalone",
  "scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "assets/icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    ...
   ]
}
```

> Asegúrate de que en la `start_url` apunte a la página de inicio correcta y despliega en modo producción en un servidor seguro con *https*. Pruébalo varias veces en distintos navegadores y verás las distintas experiencias de instalación que ofrecen.

1.2 El ngsw-worker.js

El otro gran fichero, y más importante para los desarrolladores, es **el configurador del *Service Worker***. Cuando un navegador ejecuta un *script* lo hace en *thread* dedicado a la interacción con el usuario y la manipulación del *DOM*. Normalmente ese hilo está muy ocupado, cuando no está saturado. La solución para agilizar los procesos está en usar más *threads*. Presentamos el *worker thread*.

Se le llama así a los hilos creados a partir del principal y que le ayudan en tareas en segundo plano. Esos hilos tienen **prohibido el acceso al DOM, ni lo escuchan ni lo manipulan**. Pero a cambio están muy liberados para realizar cálculos complejos o llamadas a servicios. Se comunican con el *thread* principal a través de un sencillo protocolo de eventos y subscripciones.

Una de las tareas para las que más se les utiliza es para la gestión inteligente de las comunicaciones. Este fichero viene pre programado para realizar las siguientes **funciones PWA de Angular**:

- Caché de contenido estático para funcionamiento offline
- Caché de datos dinámicos para mayor velocidad
- Gestión de instalaciones y versiones
- Notificaciones de datos push

Todo lo que hay que hacer es configurar estas funciones en un fichero, el `ngsw-config.json`, ya generado con valores por defecto. El *CLI*, durante el proceso de construcción en modo producción, copiará y manipulará los scripts y sus configuraciones. En ejecución, el `AppModule` registrará el script en el navegador, se subscribirá a sus eventos y ejecutará lo configurado por el programador en el *json*.

2 Comunicaciones y caché

El service worker de Angular está especialmente diseñado para hacerse cargo de las comunicaciones con el servidor. Digamos que se convierte en un **interceptor transparente de todas las peticiones http**. Tanto de los ficheros propios de la aplicación como de las comunicaciones de datos.

2.1 Descarga y actualización de la aplicación

Una vez descargado el *index.html* con el contenido mínimo de Angular, diríamos que la aplicación se ha instalado y está lista para ejecutarse. A partir de ese momento el *ngsw* toma el control y puede pre descargar ficheros en segundo plano; de forma que **cuando sean reclamados ya estén disponibles** y mejoren la experiencia del usuario.

Rutinariamente el servicio se ocupará de consultar novedades en el servidor para mantener los ficheros locales actualizados. Todo ello se configura en la sección `assetGroups` del `ngsw-config.json`.

Default: Full App

Se descarga la aplicación completa. Tranquilidad, esto sucede en segundo plano y una vez arrancado angular y con el usuario contento viendo ya la página pedida. Al navegar por las páginas la respuesta es instantánea porque los módulos con el código ya están ahí.

```
"assetGroups": [
  {
    "name": "app",
    "installMode": "prefetch",
    "resources": {
      "files": [
        "/favicon.ico", "/index.html",
        "/manifest.webmanifest",  "/*.css", "/*.js"
      ]
    }
  },
  {
    "name": "assets",
    "installMode": "lazy",
    "updateMode": "prefetch",
    "resources": {
      "files": [
        "/assets/**",
        "/*.(eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
      ]
    }
  }
],
```

Eso sí, la descarga consume línea. En cierto casos, con usuarios móviles y grandes aplicaciones quizás no sea adecuado. Si prefieres que los módulos Lazy no se descarguen hasta que nose visiten, te propongo esta otra configuración.

Proposed: Lazy App

```
 "assetGroups":[
   {
     "name": "coreapp",
     "installMode": "prefetch",
     "resources": {
       "files": [
         "/favicon.ico",
         "/index.html",
         "/manifest.webmanifest",
         "/*.css",
         "/common*.js",
         "/main*.js",
         "/ngsw*.js",
         "/*woker*.js",
         "/*polyfills*.js",
         "/runtime*.js"
       ]
     }
   },
   {
     "name": "lazyapp",
     "installMode": "lazy",
     "updateMode": "prefetch",
     "resources": {
       "files": ["/*.js"]
     }
   }...
```

2.2 Caché inteligente de datos

El control de la **recepción de datos dinámicos** es la otra gran tarea del *service worker*. En este caso para tener una caché que acelere la presentación de datos o que incluso permita un funcionamiento *offline*.

Mediante **dos estrategias complementarias que tratan de mantener los datos actualizados y disponibles** en todo momento. En este caso configurándolo en los `dataGroups`.

> La idea del *cache-first* es tener el dato ya listo para ser usado cuanto antes. Mientras que con *api-first* se pretende tener la versión más actual posible, y usar la última descargada en caso de problemas o desconexión total.

```
"dataGroups": [
 {
   "name": "cache-first-greeting",
   "urls": [
     "http://localhost:3333/api"
   ],
   "cacheConfig": {
     "strategy": "performance",
     "maxAge": "1d",
     "maxSize": 10
```

```
    }
  },
  {
    "name": "api-first-rates",
    "urls": [
      "https://api.exchangeratesapi.io/latest?symbols=GBP",
      "https://api.exchangeratesapi.io/latest"
    ],
    "cacheConfig": {
      "strategy": "freshness",
      "timeout": "5s",
      "maxAge": "1h",
      "maxSize": 10,
    }
  }
]
```

3 Servicios

La librería `@angular/pwa` publica el módulo `ServiceWorkerModule` que contiene la lógica de registro del *service worker* y dos servicios programables con los que interactuar desde el código de tu aplicación Angular.

3.1 Actualización con el SwUpdate

Cuando despliegas una nueva versión seguro que estás deseando que los usuarios disfruten cuanto antes de las mejoras o correcciones. Pero en las aplicaciones SPA resulta que **la actualización no es tan inmediata** como pudiera parecer. Al no forzar la recarga del documento en las navegaciones internas, el `index.html` puede residir sin cambios más de lo debido. Confiar en que lo hagan el navegador o el usuario puede no ser una opción.

La solución PWA es usar el `ServiceWorker` para detectar nuevas versiones en el servidor. Para ello utiliza un sistema propio de *hashes* que le permite comparar la versión descargada y la disponible en el servidor.** Cuando detecta un cambio emite un evento al que te puedes subscribir** usando el servicio `SwUpdate`. En ese momento puedes consultar al usuario, o forzar a lo bestia la recarga 😉.

```
constructor(private swUpdate: SwUpdate) {
  if (this.swUpdate.isEnabled) {
    this.swUpdate.available.subscribe((event: UpdateAvailableEvent) => {
      if (confirm(`Do you want to update?`)) {
        window.location.reload();
      }
    });
  }
}
```

3.2 Notificaciones con el SwPush

Una característica de las *apps* nativas muy aprecia por los usuarios es la capacidad de mostrar **mensajes recibidos vía *push* por parte del servidor**. Pueden ser avisos, novedades o simple marketing. Pero lo importante es que lo recibe el sistema operativo, la aplicación no necesita estar en marcha y el mensaje se muestra de forma nativa.

> Obviamente para ello hay que involucrar código de servidor y un servicio de mensajería de terceros. Pero la parte Angular del desarrollo es muy sencilla. Consiste en registrar al usuario, que voluntariamente decide ser notificado, y luego escuchar los mensajes provenientes en forma de eventos. Todo ello reclamando una dependencia al servicio de notificaciones `constructor(private swPush: SwPush)`.

```
constructor(private swUpdate: SwUpdate) {
  if (this.swPush.isEnabled) {
      this.swPush
        .requestSubscription({ serverPublicKey: 'VAPID_PUBLIC_KEY' })
        .then(sub => {
          console.log('send subscription to your server and wait form messages',
sub.toJSON());
          this.swPush.messages.subscribe(msg => console.log('Received: ', msg));
        })
    }
}
```

Ya tienes una web que se comporta progresivamente como una App nativa, una *Progressive Web Application*. Las mejoras en rendimiento y experiencia de usuario son recompensa más que suficiente para que le des una oportunidad a las **Angular PWA**.

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