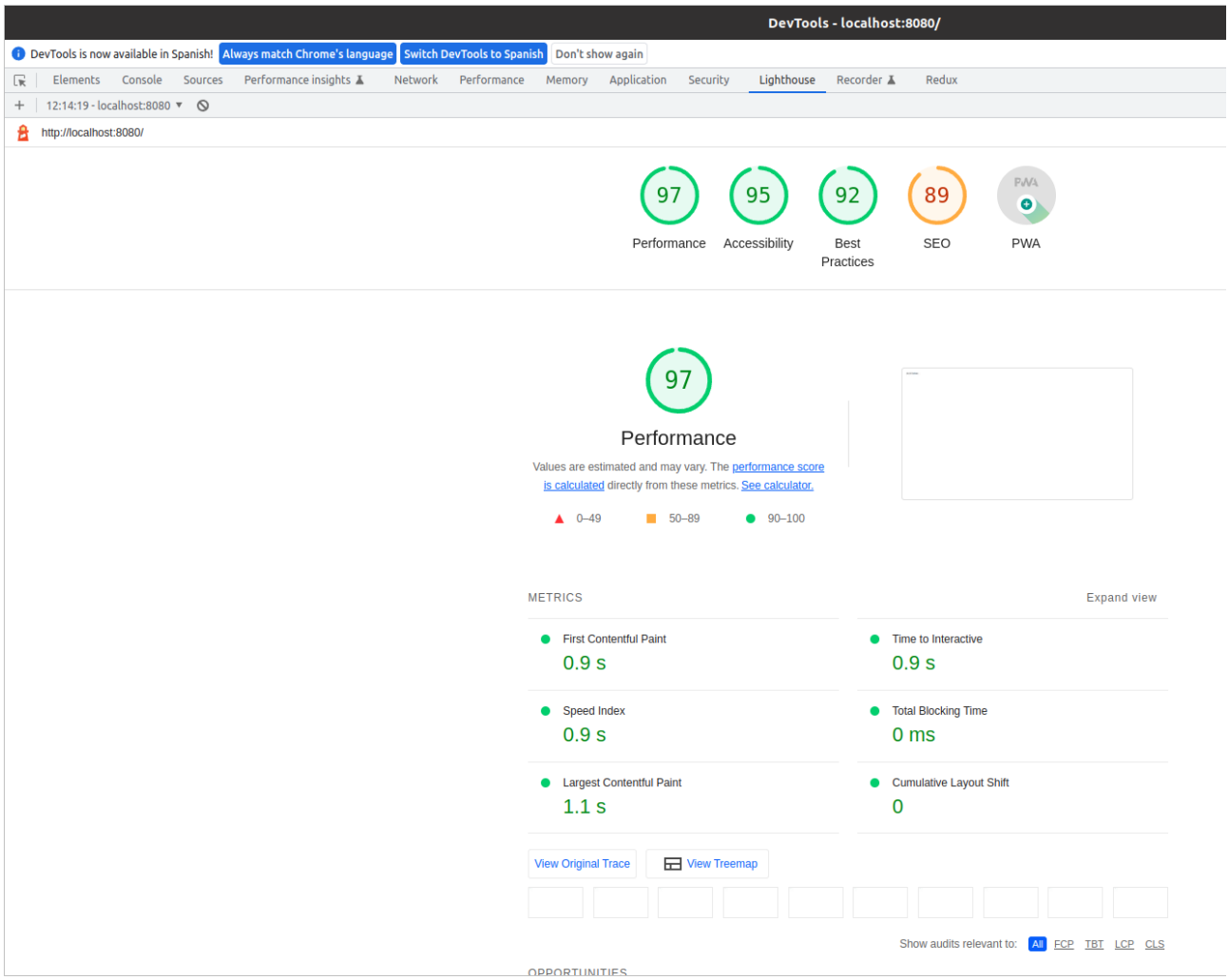
PEC5 - PWA

1. Implementar una aplicación completa libre que deberá transformarse a PWA. Configurar el manifest y todo lo necesario del service worker para que pueda instalarse y funcionar sin internet. Una vez implementada, deberemos adjuntar un pequeño documento con algunos comentarios básicos de la información que nos muestra el informe generado por la herramienta LightHouse de Google Chrome y las decisiones de diseño más relevantes en cuanto a configuración del service worker.



Comentarios básicos de la herramienta LightHouse

He realizado el estudio con los datos de LightHouse sobre la **PWA** ya implementada e instalada en el sistema y los resultados son los que aparecen en la imagen. Los resultados son muy satisfactorios debido a que la App está totalmente cacheada. El único valor que resulta desfavorable es el de SEO debido a que no son páginas estáticas, sino dinámicas por lo que dificulta la labor de SEO ya que trabajamos con plantillas; y entonces el motor de búsqueda no encuentra texto en la página lo que perjudica la puntuación en SEO.

Si las imágenes no fueran cacheadas mediante una estrategia con el service-worker la puntuación con LightHouse sería mucho más baja en todos los campos que comprende el estudio; además las imágenes, en este caso pesan mucho y no proporcionamos medidas de éstas en su contenedor a la hora de renderizarlas en la página, lo que daría una puntuación aún más baja. Por lo tanto la imagen anterior del estudio LightHouse donde los resultados son muy satisfactorios es gracias a la configuración del service-worker.

Comentarios sobre el fichero de configuración de la caché ngsw-config.json

Para implementar la App y que funcione Offline he parametrizado el fichero **ngsw-config.json** de la siguiente manera:

– En assetsGroup la clave **installmode** tiene el valor **prefetch** para que cachee todos los elementos de la página una vez cargada, y no los cachee bajo demanda (lazy) donde podría quedar algún recurso pendiente, si el usuario no lo visualiza, y no funcionar bien la App en modo offline.
En el segundo grupo de assetsGroup de nuestro fichero service-worker pertenece a las imágenes de la página y he cacheado todos los recursos de imagen mediante prefetch, y la clave **updateMode** tiene el valor lazy porque creemos que las imágenes no son recursos que se modifiquen constantemente.

– En **dataGroups** la clave **maxSize** tiene el valor de 100, donde indicamos el número máximo de urls que podemos cachear. En esto caso con 100 nos sobra. Las urls estarán en cache un máximo de una hora. La estrategia a seguir es la de **performance** de forma que el recurso primero se irá a buscar en la caché y si no lo encuentra entonces lo irá a buscar a internet. Aunque después de una hora los recursos se borran de caché, el acceso a caché es rápido y creo que no perjudica demasiado en el rendimiento nuestra estrategia aplicada en el caso de que el recurso no se encuentre en caché, de ahí que hemos puesto el valor ‘performance’.

Para que la App funcione bien Offline debemos navegar por toda la App con la finalidad de que todos los recursos se cacheen y así al instalarla en nuestro sistema todos los recursos estén disponibles en modo Offline.

Siguiendo el tutorial oficial de Angular (https://angular.io/guide/universal), adaptar la aplicación de Angular anterior para disponer de la capacidad de SSR usando schematics.

Para crear el lado servidor ejecutamos:

```
$ng add @nguniversal/express-engine
```

Al ejecutar el comando anterior se crean y actualizan los ficheros siguientes de nuestro proyecto:

Ficheros creados:

src/main.server.ts
src/app/app.server.module.ts
tsconfig.server.json
server.ts

Ficheros actualizados:

package.json //se generan scripts relacionados con SSR.
angular.json
src/main.ts
src/app/app.module.ts
src/app/app-routing.module.ts

Para lanzar la App en nuestro sistema local y **no en producción** ejecutamos el script siguiente que lanza el servidor **serve-ssr**:

```
$npm run dev:ssr
```

Construye una tabla con los ficheros creados y comenta el contenido que dispone cada uno de ellos, similar al mostrado en el tutorial.

Fichero src/main.server.ts

Este fichero contiene el código para iniciar el nodo del lado del servidor. Una vez iniciado el servidor, éste escucha las peticiones o '**requests'** del lado cliente por el puerto que toque.

La función **ngExpressEngine()** devuelve una promesa (callback) que se resuelve con la página renderizada. La página es capturada por el web server o servidor web, el cual la envía a el lado cliente.

Fichero tsconfig.server.json

```
{
        "extends": "./tsconfig.app.json",
        "compilerOptions": {
        "outDir": "./out-tsc/server",
        "target": "es2019",
        "types": [
        "node"
        ]
},
        "files": [
        "src/main.server.ts",
        "server.ts"
        ],
        "angularCompilerOptions": {
        "entryModule": "./src/app/app.server.module#AppServerModule"
        }
}
```

Fichero de configuración del lado servidor. Es una extensión del fichero tsconfig.app.json de nuestro proyecto. En él se describen la opciones de compilación para el lado servidor, en **files** tenemos los ficheros con el código para generar el lado servidor y las opciones de compilación de Angular donde el módulo de servidor **de partida** es el **app.server.module.**

Fichero src/app/app.server.module.ts

```
import { NgModule } from '@angular/core';
import { ServerModule } from '@angular/platform-server';

import { AppModule } from './app.module';
import { AppComponent } from './app.component';

@NgModule({
imports: [
AppModule,
ServerModule,
],
bootstrap: [AppComponent],
})
export class AppServerModule {}
```

En este fichero vemos que se exporta el módulo principal de la App, el AppModule, junto con el módulo de servidor. El AppServerModule es el puente entre el lado servidor Universal que renderiza y la aplicación Angular en el lado cliente.

Fichero src/main.server.ts

```
import '@angular/platform-server/init';

import { enableProdMode } from '@angular/core';

import { environment } from './environments/environment';

if (environment.production) {
enableProdMode();
}

export { AppServerModule } from './app/app.server.module';
export { renderModule } from '@angular/platform-server';
```

Este código **inicializa el servidor** dependiendo del entorno donde se va a ejecutar la App (desarrollo, producción).

Url de PWA deployada en Netlify:

https://avanzados-pec3.netlify.app/