

TEMA 4

Progressive Web App

Desarrollo front-end avanzado

**Máster Universitario en Desarrollo de
sitios y aplicaciones web**



**Universitat Oberta
de Catalunya**



1. Introducción

En este tema estudiaremos las Progressive Web Apps (PWAs), un enfoque moderno que permite que las aplicaciones web se comporten de manera similar a una aplicación nativa: pueden funcionar sin conexión, instalarse en dispositivos, mostrar iconos propios, mejorar la velocidad de carga y ofrecer una experiencia de usuario fluida y consistente.

La relevancia de las PWAs en el entorno actual es enorme:

- Han sido adoptadas por grandes aplicaciones como Twitter Lite, Pinterest, Telegram Web o Spotify Web.
- Permiten desarrollar una única base de código para escritorio, móvil y tablet.
- Reducen costes respecto al desarrollo nativo (Android + iOS).
- Funcionan incluso en dispositivos de bajo rendimiento.

En este tema aprenderemos:

- Qué es exactamente una PWA.
- Qué tecnologías utiliza.
- Cómo implementarla correctamente con Angular 19.
- Cómo configurar caché estático y dinámico mediante service workers.
- Cómo probar la aplicación offline.
- Cómo desplegarla en GitHub Pages, haciéndola accesible desde cualquier dispositivo.

2. ¿Qué es una Progressive Web App?

Una Progressive Web App es una aplicación web que combina lo mejor del mundo web y lo mejor del mundo móvil. Su objetivo es ser progresiva, es decir, aprovechar capacidades avanzadas de los navegadores modernos sin dejar de ser completamente funcional en navegadores antiguos.

Una PWA debe cumplir las siguientes propiedades:

- Progresiva: funciona para todos los usuarios.
- Adaptable: se ajusta a cualquier factor de forma (móvil, tablet, desktop...).
- Offline-first: funciona sin conexión o con conexión limitada.
- Estilo app: interfaz similar a una app nativa.
- Actualizable: siempre sirve la versión más reciente gracias a la caché controlada.
- Segura: solo funciona en entornos HTTPS.
- Instalable: permite instalarse en el dispositivo dando lugar a una “app” web instalada.
- Vinculable: accesible mediante URL, sin tiendas de apps y sin procesos de instalación complejos.

Estas capacidades se consiguen mediante una serie de tecnologías que estudiaremos a continuación.

3. Tecnologías clave en una PWA

3.1 Service Worker

El Service Worker es el corazón de cualquier PWA moderna. Se trata de un script que se ejecuta en segundo plano y actúa como proxy entre la aplicación y la red.

Responsabilidades principales:

- Interceptar peticiones HTTP.
- Servir contenido desde caché.
- Aplicar estrategias de recuperación de datos offline.
- Gestionar la instalación y el ciclo de vida de la app.
- Mejorar el rendimiento cargando recursos desde caché.

Un Service Worker no tiene acceso directo al DOM, por lo que la comunicación se realiza mediante mensajes (postMessage).

3.2. Manifest

El manifest.webmanifest describe:

- Iconos de instalación
- Nombre corto y largo de la aplicación
- Tema de la app (color de cabecera, color de fondo)
- Orientación
- Pantalla de inicio (standalone, fullscreen, etc.)

Cuando el navegador detecta un manifest válido y un service worker funcionando, mostrará el prompt de instalación.

3.3. Estrategias de caché

Una PWA puede decidir cómo recuperar los recursos:

- cache-first: primero intenta caché, si no existe va a la red.
- network-first: primero va a la red, si falla utiliza caché.
- stale-while-revalidate: usa caché y actualiza en segundo plano.
- freshness (Angular): intenta primero la red y si no responde en un tiempo límite, usa caché.

Estas estrategias se configuran en ngsw-config.json.

3.4. App Shell

El App Shell es la estructura mínima de la aplicación:

- HTML base
- CSS esenciales
- Framework (Angular)
- Componentes estructurales

Esto permite que la app cargue instantáneamente incluso sin conexión.

3.5. Angular Service Worker

Angular proporciona su propio service worker, instalado mediante:

```
ng add @angular/pwa
```

Esto genera:

- ngsw-config.json
- manifest optimizado
- iconos
- scripts en Angular para registrar el SW

Nota técnica sobre Angular Service Worker (Angular 16–19)

A partir de Angular 16, el paquete oficial `@angular/service-worker` pasó a un estado de mantenimiento. Esto significa que continúa siendo plenamente funcional, estable y compatible con las versiones actuales del framework, incluyendo Angular 19, pero ya no se incorporarán nuevas características.

Es importante destacar que no está deprecado, no ha sido eliminado y sigue siendo la solución oficial para crear PWAs con Angular. La decisión del equipo de Angular responde a que la API de Service Workers es un estándar web maduro que apenas ha cambiado en los últimos años.

Las capacidades fundamentales de una PWA —como el funcionamiento offline, el almacenamiento en caché, la instalación en dispositivos, las actualizaciones en segundo plano o la arquitectura App Shell— forman parte de los estándares del navegador (Service Worker API, Cache API, Web App Manifest). Por ello, el estudio de PWAs continúa siendo totalmente vigente y relevante para comprender cómo funcionan las aplicaciones web modernas en entornos multiplataforma.

En consecuencia, aunque Angular Service Worker ya no reciba nuevas funcionalidades, sigue siendo una herramienta válida y recomendada para el desarrollo de PWAs con Angular, especialmente en contextos educativos y en aplicaciones donde el comportamiento offline y la experiencia tipo app sean requisitos importantes.

3.6. Testing de una PWA en local con http-server

El Service Worker no funciona en ng serve.

Necesitamos:

```
ng build  
npm install -g http-server  
http-server -p 8080 ./dist/pwa-demo
```

A partir de ese momento podemos:

- Inspeccionar el service worker
- Activar el modo offline
- Ver el comportamiento de la app sin conexión

Todos estos conceptos los iremos trabajando en el proyecto guiado paso a paso del siguiente apartado.

4. Ejemplo guiado: PWA desde cero

Después de haber visto un poco de teoría, desarrollaremos nuestra propia **PWA**.

Desarrollaremos un ejemplo sencillo, para tener una visión inicial de las posibilidades que ofrece la tecnología **PWA**. Un punto interesante que quedaría por tratar, pero que queda fuera del alcance de este temario, sería la gestión de notificaciones **push**, lo cual recomendamos que indaguéis un poco por vuestra cuenta.

Con todo esto, lo primero que haremos será crearnos un proyecto nuevo e implementaremos una vista maestro detalle utilizando la api que vimos en el tema 3, concretamente:

<https://picsum.photos/>

Se trata de una API pública que podemos utilizar para devolver un listado de imágenes, una imagen concreta en función de un identificador, ...

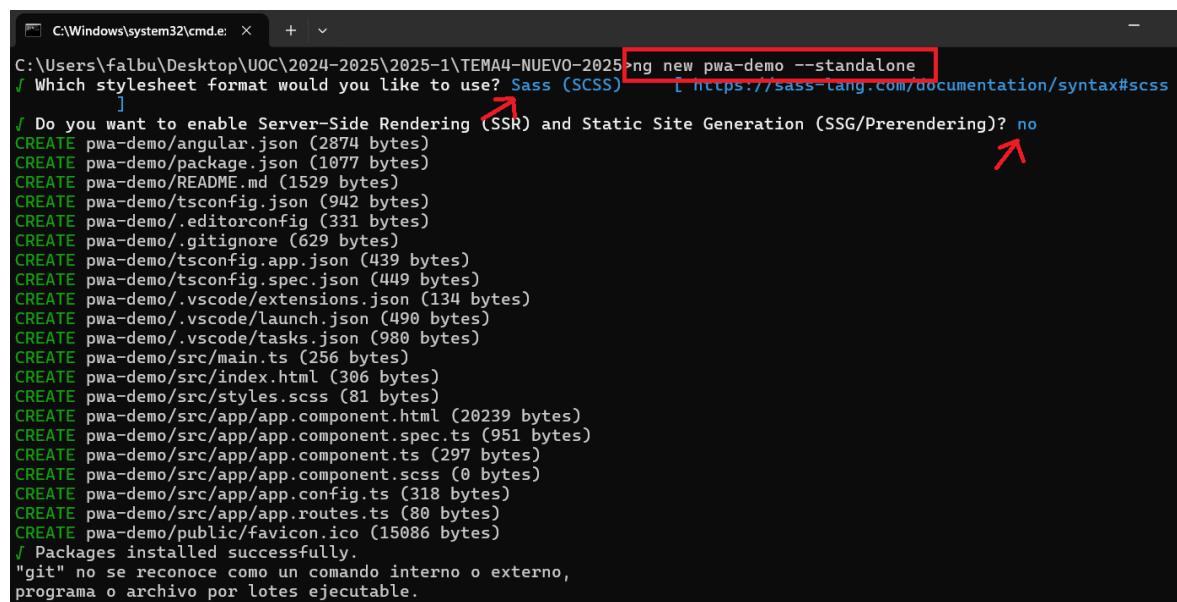
En nuestro contexto, suficiente para implementar una aplicación sencilla con la típica vista lista-detalle y poder estudiar una introducción a cómo transformar esta aplicación a **PWA**.

A continuación, enumeramos una propuesta de paso a paso a seguir:

- **Paso 1:**

Creamos un nuevo proyecto ejecutando el siguiente comando desde la consola:

Nota: En este caso de ejemplo se trabaja con la **versión Angular 19**.



```
C:\Windows\system32\cmd.exe: x + ^ C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025>ng new pwa-demo --standalone
? Which stylesheet format would you like to use? Sass (SCSS) [ https://sass-lang.com/documentation/syntax#scss ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? no
CREATE pwa-demo/angular.json (2874 bytes)
CREATE pwa-demo/package.json (1077 bytes)
CREATE pwa-demo/README.md (1529 bytes)
CREATE pwa-demo/tsconfig.json (942 bytes)
CREATE pwa-demo/.editorconfig (331 bytes)
CREATE pwa-demo/.gitignore (629 bytes)
CREATE pwa-demo/tsconfig.app.json (439 bytes)
CREATE pwa-demo/tsconfig.spec.json (449 bytes)
CREATE pwa-demo/.vscode/extensions.json (134 bytes)
CREATE pwa-demo/.vscode/launch.json (490 bytes)
CREATE pwa-demo/.vscode/tasks.json (980 bytes)
CREATE pwa-demo/src/main.ts (256 bytes)
CREATE pwa-demo/src/index.html (306 bytes)
CREATE pwa-demo/src/styles.scss (81 bytes)
CREATE pwa-demo/src/app/app.component.html (20239 bytes)
CREATE pwa-demo/src/app/app.component.spec.ts (951 bytes)
CREATE pwa-demo/src/app/app.component.ts (297 bytes)
CREATE pwa-demo/src/app/app.component.scss (0 bytes)
CREATE pwa-demo/src/app/app.config.ts (318 bytes)
CREATE pwa-demo/src/app/app.routes.ts (80 bytes)
CREATE pwa-demo/public/favicon.ico (15086 bytes)
? Packages installed successfully.
"git" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
```

Podemos indicar que utilizaremos SCSS como formato de estilos. Por otro lado, le decimos que no habilitamos SSR y SSG.

Ejecutamos el proyecto con la instrucción:

```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng serve --open
Component HMR has been enabled, see https://angular.dev/hmr for more info.

Initial chunk files | Names | Raw size
polyfills.js | polyfills | 89.77 kB |
main.js | main | 47.89 kB |
styles.css | styles | 96 bytes |

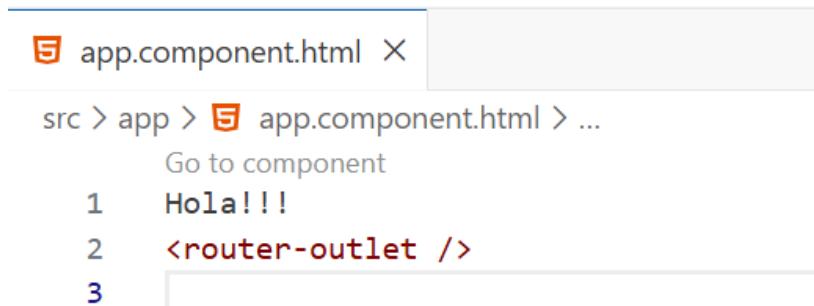
| Initial total | 137.76 kB

Application bundle generation complete. [1.529 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
→ Local: http://localhost:4200/
→ press h + enter to show help
```

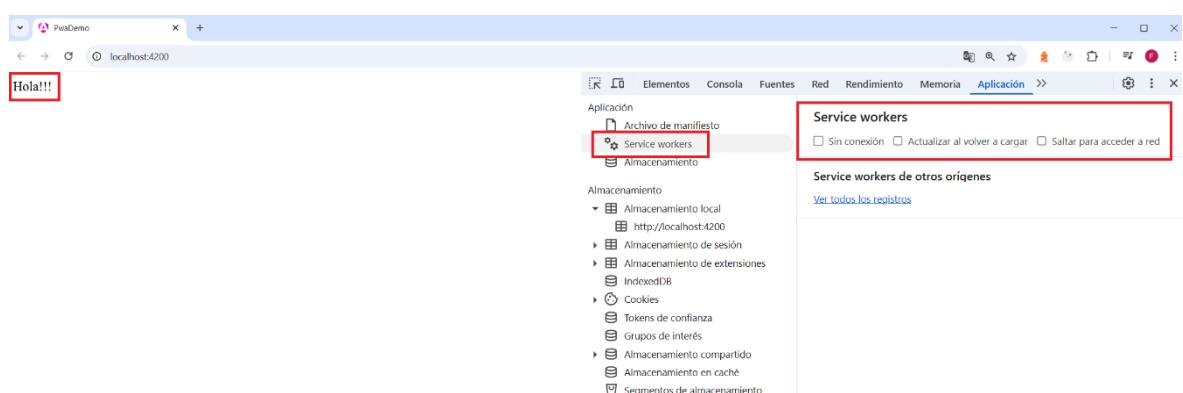
Y validamos que vemos la pantalla inicial del proyecto en el navegador.

Veremos la pantalla inicial con mucha información de **Angular** que en principio no nos interesa, así que iremos al fichero **app.component.html** y eliminaremos todo su contenido a excepción de la línea del **router-outlet** y, además, pongamos algún texto, simplemente para que se muestre alguna cosa para mostrar el modo **offline**:



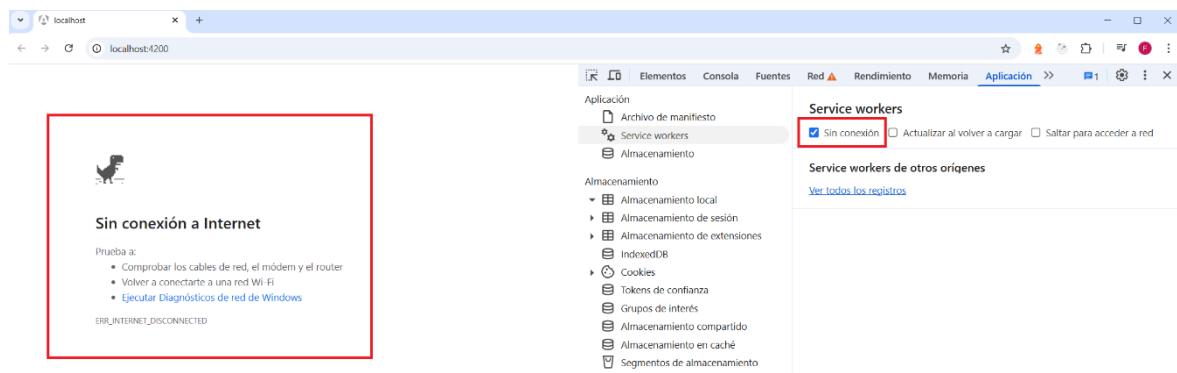
```
5 app.component.html X
src > app > 5 app.component.html > ...
Go to component
1 Hola!!!
2 <router-outlet />
3
```

Ahora en el navegador veríamos:



Veremos el mensaje en el navegador y podemos ver en la pestaña **Application** (si pulsamos F12 en el **Google Chrome**) como por ahora, no se gestiona ningún **service worker** como es normal, ya que hasta el momento tenemos una aplicación **Angular** normal. (Tampoco tenemos ningún **manifest**)

Hagamos una prueba, si pulsamos el botón de **offline**, para simular que no tenemos conexión a internet y refrescamos, ¿qué pasará?



Pues que la aplicación dejará de funcionar ya que no tiene conexión a internet. Posteriormente, cuando tengamos la **PWA** configurada, podremos observar cómo, aunque no tengamos conexión a internet, la aplicación continuará funcionando ya que trabajará con la caché, esto lo gestionaremos mediante el **service worker**.

● Paso 2:

Ahora lo que haremos es añadir algunas bibliotecas a nuestro proyecto para entender cómo gestionar la cache una vez tengamos configurada nuestra **PWA**. Nos vincularemos una librería **dinámica** y otra **estática** para entender cómo gestionarlas mediante una **PWA**.

Por ahora, lo que haremos será añadir los estilos de **Bootstrap** directamente a nuestro **index.html**, de esta manera, tendremos una librería **dinámica**.

Podemos ir a la página de **Bootstrap**:

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

Y copiar directamente los estilos usando el CDN:

The screenshot shows the Bootstrap v5.0 documentation page. On the left, there's a sidebar with navigation links like 'Getting started' (Introduction, Download, Contents, Browsers & devices, JavaScript, Build tools, Webpack, Accessibility, RFS, RTL), 'Customize' (Layout, Content, Forms, Components, Helpers, Utilities, Extend, About), and 'CSS'. The main content area has a large heading 'Introduction' with a sub-section 'Quick start' containing text about adding Bootstrap via jsDelivr or downloading source files. Below that is a 'CSS' section with the text 'Copy-paste the stylesheet <link> into your <head> before all other stylesheets to load our CSS.' A code snippet is shown: <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet". A 'Copy' button is highlighted with a red box. To the right, there's a sidebar titled 'On this page' with links to various Bootstrap components and resources.

Y pegarlos en nuestro **index.html**:

The screenshot shows a code editor with an open file named 'index.html'. The code is as follows:

```
index.html
src > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
5      <title>PwaDemo</title>
6      <base href="/" />
7      <meta name="viewport" content="width=device-width, initial-scale=1" />
8      <link rel="icon" type="image/x-icon" href="favicon.ico" />
9      <link
10        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"
11        rel="stylesheet"
12        integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOMLASjC"
13        crossorigin="anonymous"
14      />
15    </head>
16    <body>
17      <app-root></app-root>
18    </body>
19  </html>
```

A specific line of code is highlighted with a red box, containing the Bootstrap CSS link: <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet".

Las propiedades **integrity** y **crossorigin** podemos quitarlas.

Ahora que tenemos enlazada una biblioteca de estilos a nuestro fichero **index.html** y, por lo tanto, la tendremos que cargar/gestionar de manera **dinámica**, vamos a descargarnos otra biblioteca de estilos directamente a nuestro proyecto para de esta manera, tener una biblioteca **estática**, es decir, una biblioteca que tenemos en nuestro propio proyecto, que no será necesario salir a pedirla a internet.

Aquí tenemos muchas posibilidades, por ejemplo, podemos ir a la url:

<https://animate.style/>

Y en el apartado de *instalación y uso*:

Installation and usage

Installing

Install with npm:

```
$ npm install animate.css --save
```

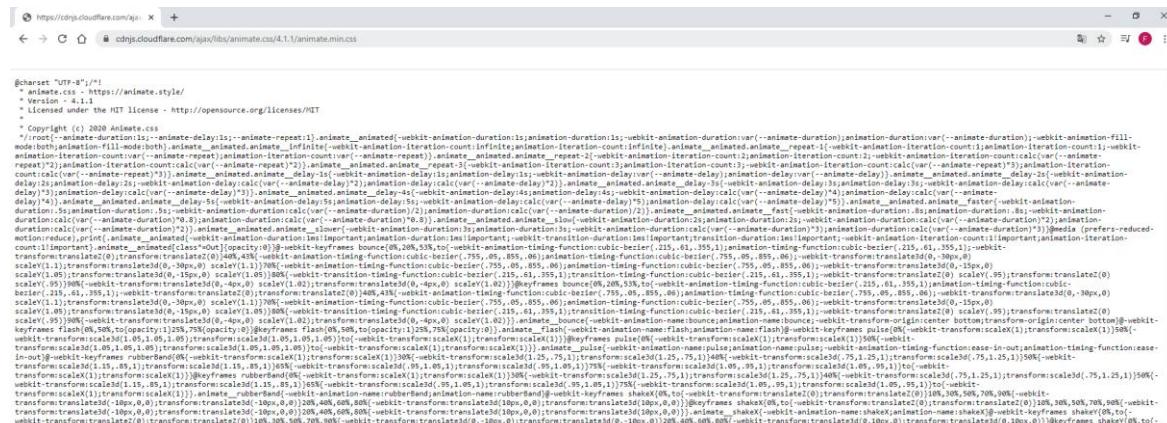
with yarn:

```
$ yarn add animate.css
```

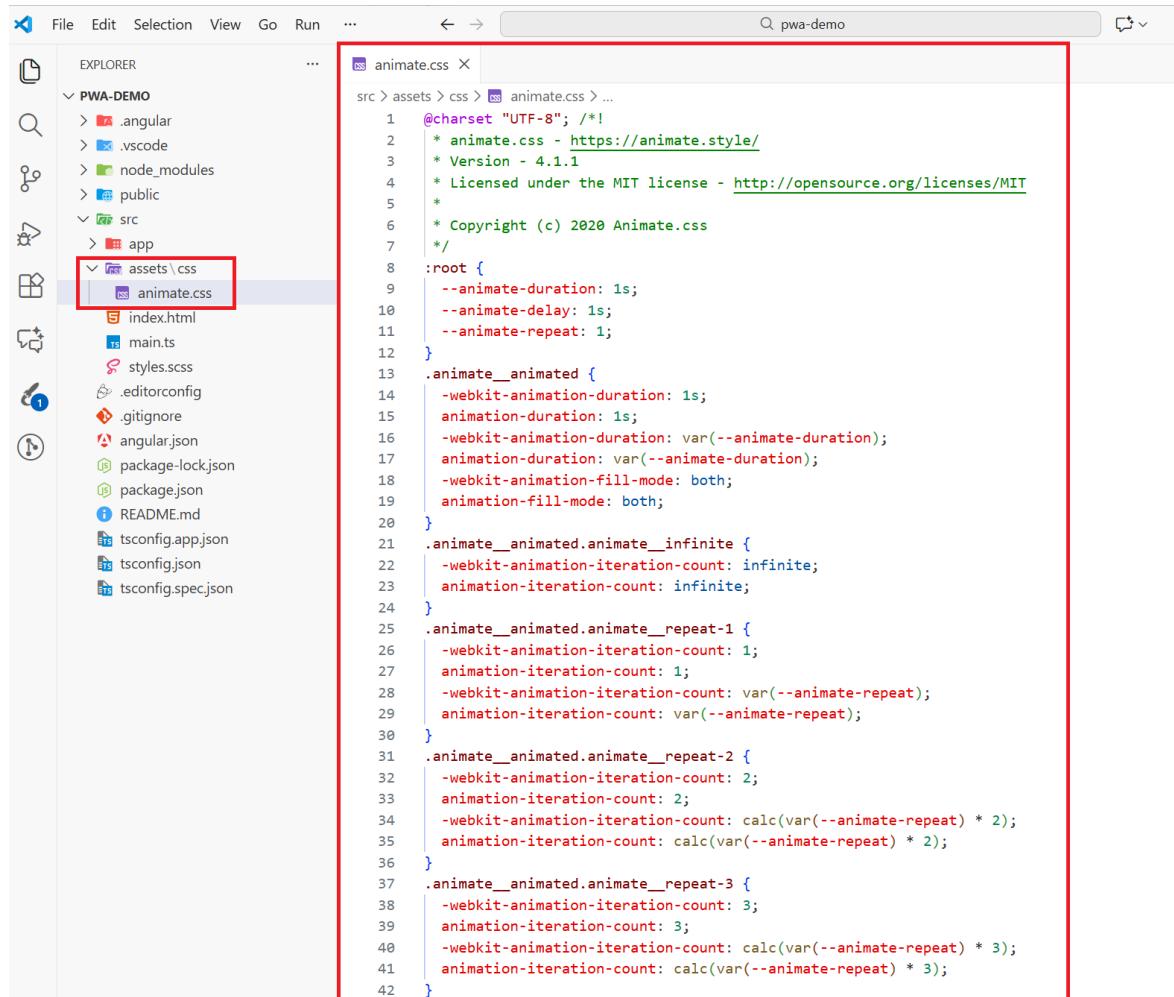
or add it directly to your webpage using a CDN:

```
<head>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"/>
</head>
```

Podemos copiar directamente la **url** que está marcada en rojo en el navegador para ver el código completo de la biblioteca:



Copiaremos todo el código de esta librería a nuestro proyecto de la siguiente manera:

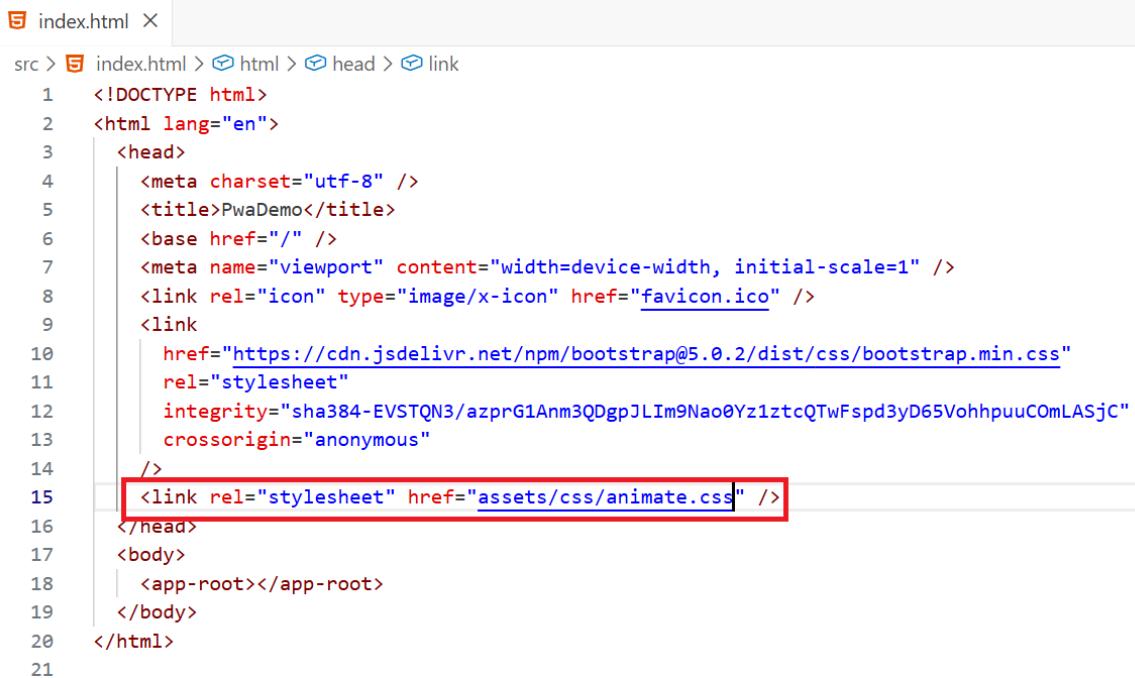


The screenshot shows a Windows File Explorer window with a red border. Inside, the file structure of a 'PWA-DEMO' project is visible. In the 'assets\css' folder, there is a file named 'animate.css'. This file is highlighted with a red box. The code editor on the right displays the contents of 'animate.css', which is a CSS file containing various animation rules for the Animate.css library.

```
src > assets > css > animate.css > ...
1  @charset "UTF-8"; /*!
2  * animate.css - https://animate.style/
3  * Version - 4.1.1
4  * Licensed under the MIT license - http://opensource.org/licenses/MIT
5  *
6  * Copyright (c) 2020 Animate.css
7  */
8 :root {
9   --animate-duration: 1s;
10  --animate-delay: 1s;
11  --animate-repeat: 1;
12 }
13 .animate__animated {
14   -webkit-animation-duration: 1s;
15   animation-duration: 1s;
16   -webkit-animation-duration: var(--animate-duration);
17   animation-duration: var(--animate-duration);
18   -webkit-animation-fill-mode: both;
19   animation-fill-mode: both;
20 }
21 .animate__animated.animate__infinite {
22   -webkit-animation-iteration-count: infinite;
23   animation-iteration-count: infinite;
24 }
25 .animate__animated.animate__repeat-1 {
26   -webkit-animation-iteration-count: 1;
27   animation-iteration-count: 1;
28   -webkit-animation-iteration-count: var(--animate-repeat);
29   animation-iteration-count: var(--animate-repeat);
30 }
31 .animate__animated.animate__repeat-2 {
32   -webkit-animation-iteration-count: 2;
33   animation-iteration-count: 2;
34   -webkit-animation-iteration-count: calc(var(--animate-repeat) * 2);
35   animation-iteration-count: calc(var(--animate-repeat) * 2);
36 }
37 .animate__animated.animate__repeat-3 {
38   -webkit-animation-iteration-count: 3;
39   animation-iteration-count: 3;
40   -webkit-animation-iteration-count: calc(var(--animate-repeat) * 3);
41   animation-iteration-count: calc(var(--animate-repeat) * 3);
42 }
```

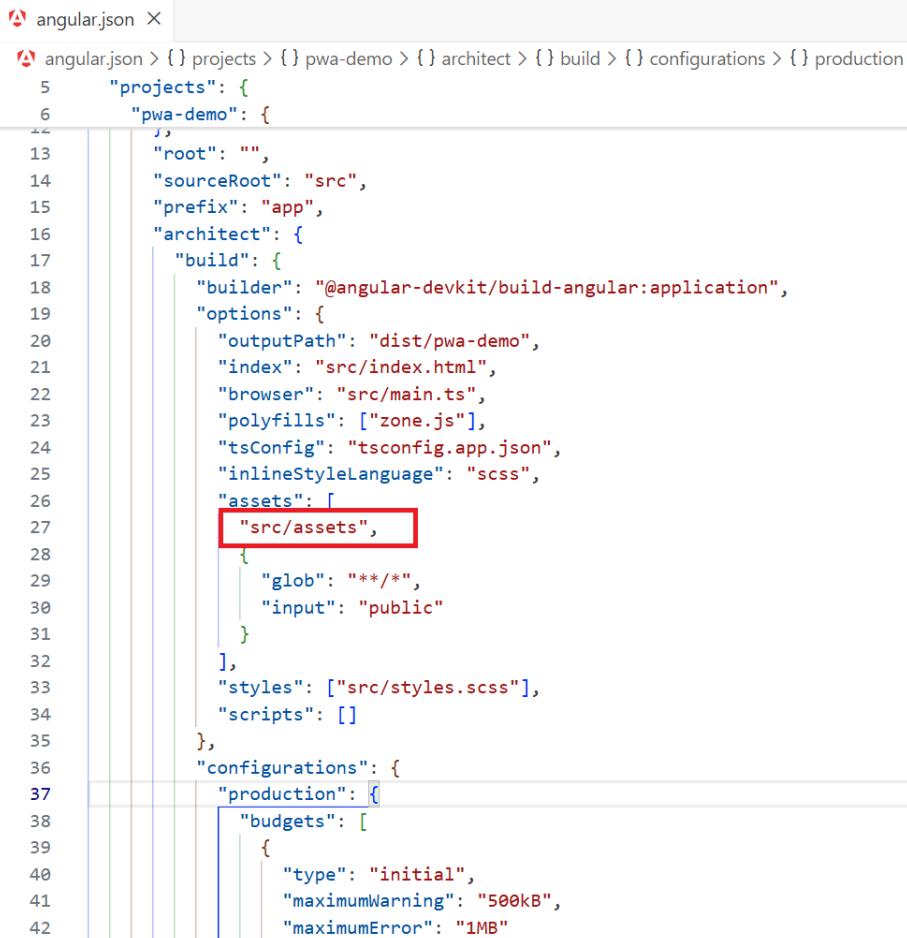
Dentro de la carpeta **assets** creamos una carpeta **css** y dentro un fichero **animate.css** donde pegaremos el contenido de la biblioteca. Podemos hacer clic derecho en el propio fichero **animate.css** una vez el código copiado y dar a **Format document** para que nos formatee el fichero ya que viene en formato reducido, pero no es importante esto, simplemente queremos una biblioteca en nuestro propio proyecto para manejarla de manera **estática**.

Por lo tanto, lo último que nos quedaría es vincular a nuestro **index.html** esta biblioteca estática:



```
index.html
src > index.html > html > head > link
1  <!DOCTYPE html>
2  <html lang="en">
3  |  <head>
4  |  |  <meta charset="utf-8" />
5  |  |  <title>PwaDemo</title>
6  |  |  <base href="/" />
7  |  |  <meta name="viewport" content="width=device-width, initial-scale=1" />
8  |  |  <link rel="icon" type="image/x-icon" href="favicon.ico" />
9  |  |  <link
10 |  |  |  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
11 |  |  |  rel="stylesheet"
12 |  |  |  integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWspd3yD65VohhpuuCOMLASjc"
13 |  |  |  crossorigin="anonymous"
14 |  |  />
15 |  |  <link rel="stylesheet" href="assets/css/animate.css" />
16 |  |  </head>
17 |  |  <body>
18 |  |  |  <app-root></app-root>
19 |  |  |  </body>
20 |  |  </html>
21
```

Y añadimos la carpeta assets creada al fichero de configuración de angular:



```
angular.json
angular.json > {} projects > {} pwa-demo > {} architect > {} build > {} configurations > {} production
5  "projects": {
6    "pwa-demo": {
12    },
13    "root": "",
14    "sourceRoot": "src",
15    "prefix": "app",
16    "architect": {
17      "build": {
18        "builder": "@angular-devkit/build-angular:application",
19        "options": {
20          "outputPath": "dist/pwa-demo",
21          "index": "src/index.html",
22          "browser": "src/main.ts",
23          "polyfills": ["zone.js"],
24          "tsConfig": "tsconfig.app.json",
25          "inlineStyleLanguage": "scss",
26          "assets": [
27            "src/assets",
28            {
29              "glob": "**/*",
30              "input": "public"
31            }
32          ],
33          "styles": ["src/styles.scss"],
34          "scripts": []
35        },
36        "configurations": {
37          "production": {
38            "budgets": [
39              {
40                "type": "initial",
41                "maximumWarning": "500kB",
42                "maximumError": "1MB"
43              }
44            ]
45          }
46        }
47      }
48    }
49  }
```

● Paso 3:

Vamos a crear los componentes iniciales.

```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng g component pages/photos-list --standalone --skip-tests
CREATE src/app/pages/photos-list/photos-list.component.html (27 bytes)
CREATE src/app/pages/photos-list/photos-list.component.ts (245 bytes)
CREATE src/app/pages/photos-list/photos-list.component.scss (0 bytes)
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng g component pages/photos-detail --standalone --skip-tests
CREATE src/app/pages/photos-detail/photos-detail.component.html (29 bytes)
CREATE src/app/pages/photos-detail/photos-detail.component.ts (253 bytes)
CREATE src/app/pages/photos-detail/photos-detail.component.scss (0 bytes)
```

El componente **photos-list** nos servirá para mostrar el listado de imágenes y el componente **photos-detail** nos servirá para mostrar el detalle de cada una de las imágenes.

● Paso 4:

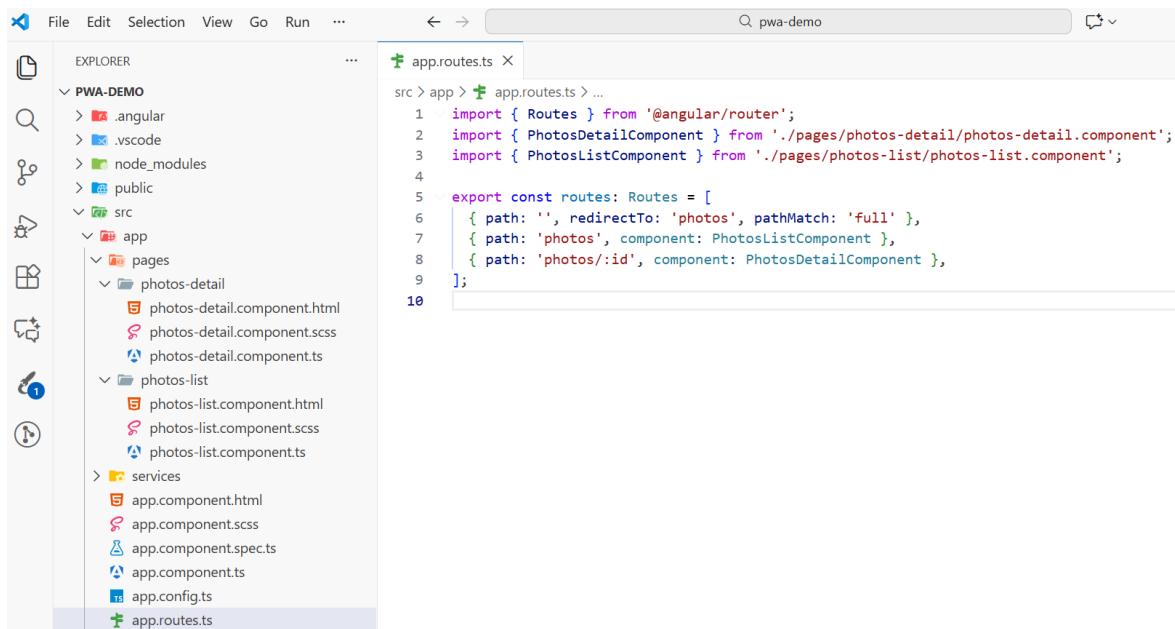
Vamos a crearnos el servicio inicial.

Ahora nos crearemos el servicio para poder llamar a la API:

```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng g service services/photos
● CREATE src/app/services/photos.service.spec.ts (373 bytes)
CREATE src/app/services/photos.service.ts (144 bytes)
```

● Paso 5:

Vamos a crearnos las rutas de navegación necesarias:

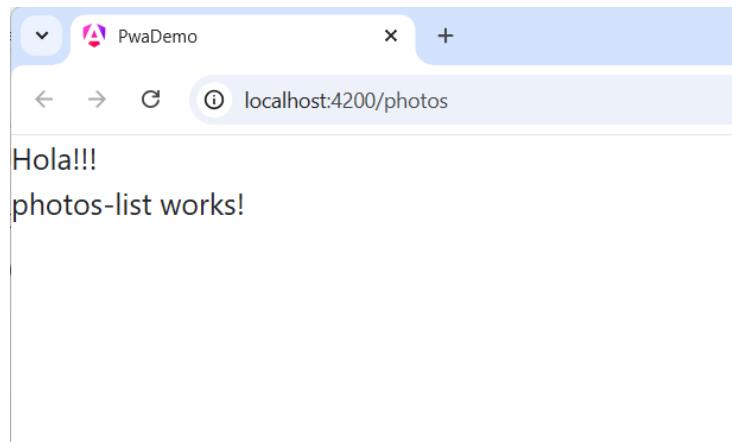


Básicamente le diremos que la ruta vacía redireccionará al listado de imágenes.

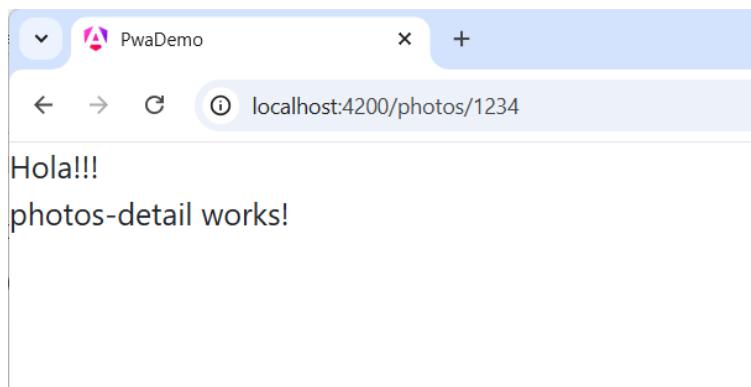
Por otro lado, para el detalle de una imagen definiremos una ruta en la que le pasaremos por ruta el **id** de la imagen determinada y redireccionará al componente **PhotosDetailComponent**.

En cualquier otro caso, que redirccione al listado de imágenes.

Si ejecutamos nuestra aplicación deberíamos ver en el navegador algo así:



Si le pasamos por ejemplo por ruta **/photos/1234** y cualquier cosa como identificador deberíamos ver algo así:



Nótese que vemos que hemos accedido al componente del detalle, por lo tanto, apuntamos al componente correcto.

- **Paso 6:**

Empecemos a implementar el **photos-list.component.html**:



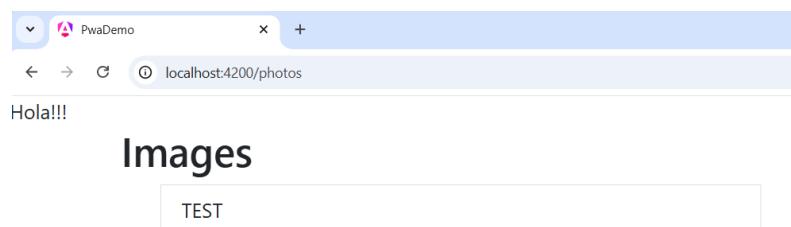
```
photos-list.component.html <input type="button" value="X">
src > app > pages > photos-list > photos-list.component.html > ...
Go to component
1   <div class="container">
2     <h1>Images</h1>
3     <ul class="list-grup">
4       <a
5         [routerLink]="/photos", 'XXXX'
6         class="list-group-item list-group-item-action animate__animated animate__bounce animate__fadeIn"
7       >
8         TEST
9       </a>
10      </ul>
11    </div>
```

Creamos un contenedor que contendrá un listado de imágenes, posteriormente esto se cargará con los datos de la API, pero de momento, nos montamos la estructura.

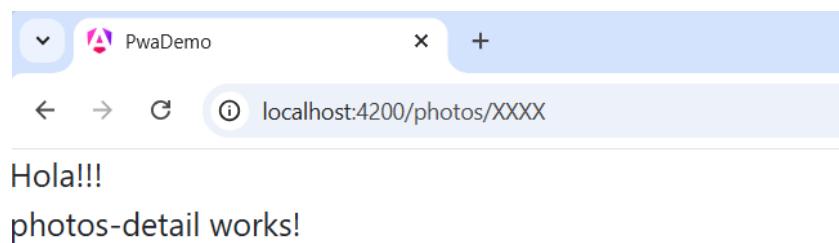
Podemos ver que en cada elemento del listado le hemos añadido las clases **animate__animated animate__bounce animate__fadeIn** para que cada elemento de la lista tenga una pequeña animación. Esto nos servirá para posteriormente ver que aun no teniendo conexión a internet nos funcionará correctamente. Esta biblioteca recordemos que la tenemos en nuestro propio proyecto.

Por otra parte, vemos que cada elemento de la lista que posteriormente contendrá el nombre de cada imagen tiene el **routerLink** para ir a la vista detallada, vemos que le pasaremos dinámicamente el identificador de cada imagen, ahora por el momento, le pasamos la cadena **XXXX** solo para validar que la navegación es correcta.

Con esto, podemos ver que inicialmente tenemos:



Solo tenemos un elemento en la lista de momento, pero podremos observar cómo aparece con una pequeña animación, además, si hacemos clic en este elemento, navegaremos a su vista detalle:



Vemos que navegamos correctamente a la vista detalle, es decir, al componente **photos-detail.component.html** y le pasamos por argumento la cadena **XXXX** que posteriormente será el identificador de la imagen.

- **Paso 7:**

Implementemos el servicio **photos.service.ts**.

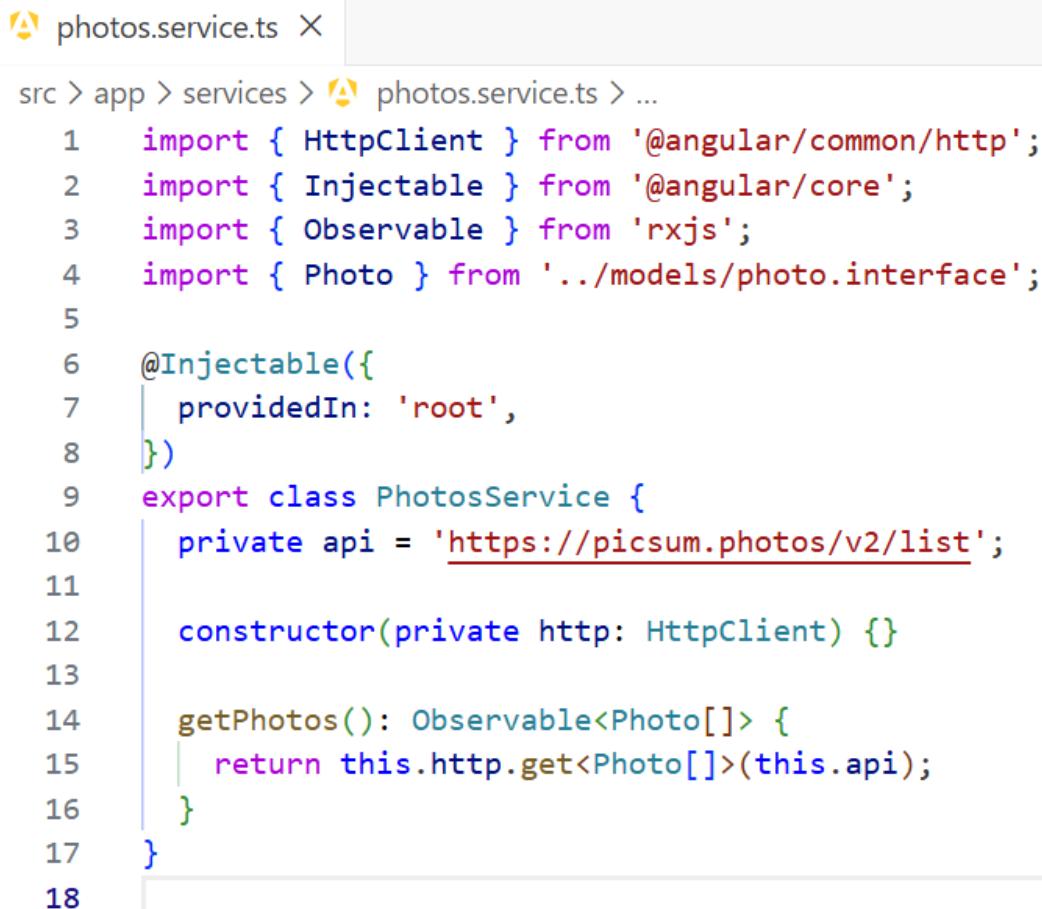
Primero creamos la interfaz para mapear la respuesta de la api de la entidad **Photo**.

A screenshot of the Visual Studio Code interface. The left sidebar shows the file structure of a project named 'PWA-DEMO'. In the 'src/app/models' folder, there is a file named 'photo.interface.ts'. This file contains the following TypeScript code, which is highlighted with a red box:

```
ts photo.interface.ts X
src > app > models > ts photo.interface.ts > ...
1 export interface Photo {
2   id: string;
3   author: string;
4   width: number;
5   height: number;
6   url: string;
7   download_url: string;
8 }
```

Nos podemos crear una carpeta **models** y dentro el fichero **photo.interface.ts** con el código que mostramos en la imagen anterior.

Implementemos el servicio:



The screenshot shows a code editor window with the file 'photos.service.ts' open. The file is located in the 'src/app/services' directory. The code implements a service to get photos from the PicSum API.

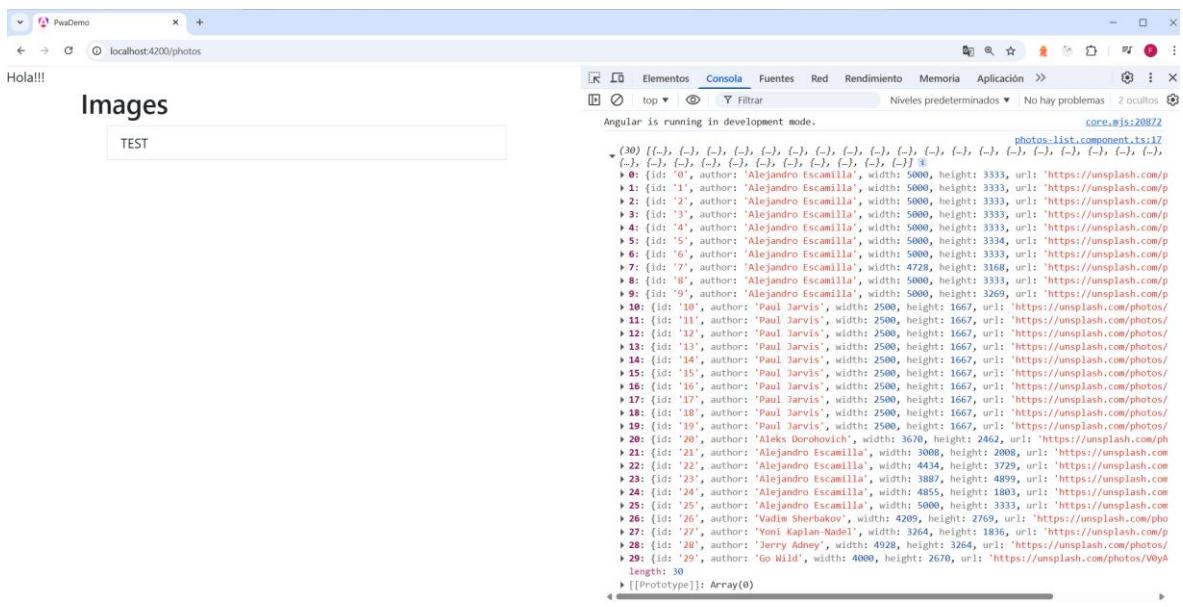
```
src > app > services > photos.service.ts > ...
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { Photo } from '../models/photo.interface';
5
6 @Injectable({
7   providedIn: 'root',
8 })
9 export class PhotosService {
10   private api = 'https://picsum.photos/v2/list';
11
12   constructor(private http: HttpClient) {}
13
14   getPhotos(): Observable<Photo[]> {
15     return this.http.get<Photo[]>(this.api);
16   }
17 }
18
```

● Paso 8:

Llamemos al servicio del componente **PhotosListComponent**:

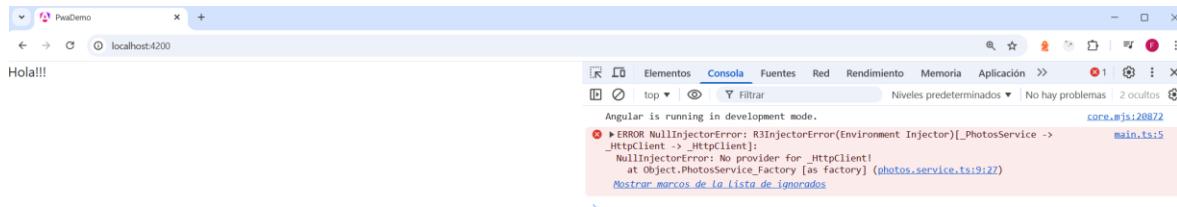
```
photos-list.component.ts X
src > app > pages > photos-list > photos-list.component.ts > ...
1 import { Component, signal } from '@angular/core';
2 import { RouterLink } from '@angular/router';
3 import { Photo } from '../../../../../models/photo.interface';
4 import { PhotosService } from '../../../../../services/photos.service';
5
6 @Component({
7   selector: 'app-photos-list',
8   imports: [RouterLink],
9   templateUrl: './photos-list.component.html',
10  styleUrls: ['./photos-list.component.scss'],
11 })
12 export class PhotosListComponent {
13   photos = signal<Photo[]>([[]]);
14   constructor(private photosService: PhotosService) {}
15
16   ngOnInit(): void {
17     this.photosService.getPhotos().subscribe(res => console.log(res));
18   }
19 }
```

Observemos el navegador:

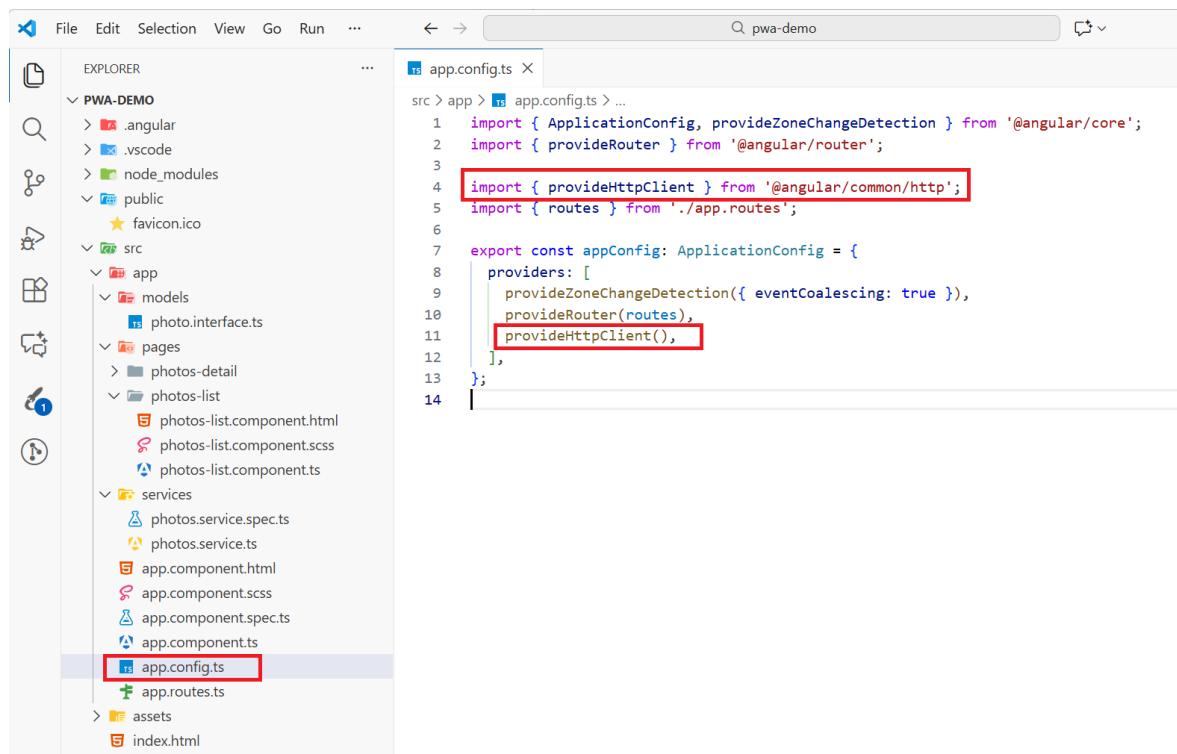


Nota! Posible error

Si nos aparece el siguiente error:



Es porque no has registrado el `HttpClient` en el “árbol” de inyección de dependencias de Angular. Por lo tanto, registraremos el `HttpClient` en `appConfig` de la siguiente manera:



Con esto ya podremos ejecutar la aplicación y ver la respuesta de la api en la consola del navegador. Es un buen momento para inspeccionar la llamada a la api des de la pestaña “network” o “red” del navegador.

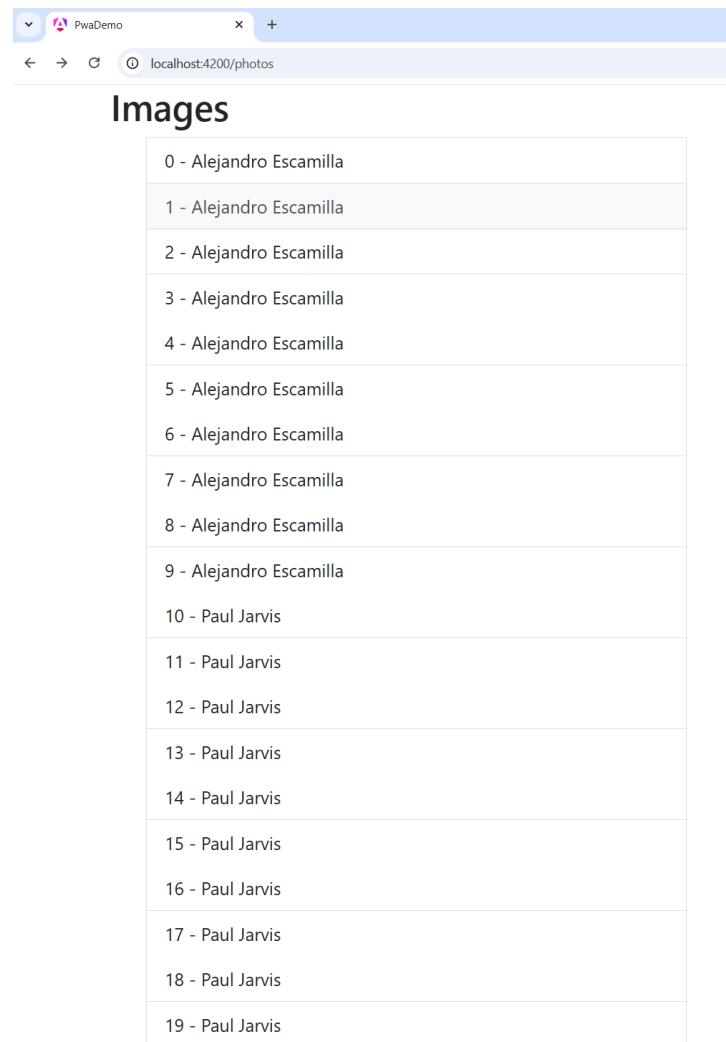
Ahora vamos a mapear la respuesta de esta llamada a una variable pública para mostrar los datos en el listado de imágenes en el fichero **photos-list.component.html**. Primero modifiquemos un poco el componente **.ts**:

```
photos-list.component.ts X
src > app > pages > photos-list > photos-list.component.ts > ...
1 import { Component, signal } from '@angular/core';
2 import { RouterLink } from '@angular/router';
3 import { Photo } from '../../../../../models/photo.interface';
4 import { PhotosService } from '../../../../../services/photos.service';
5
6 @Component({
7   selector: 'app-photos-list',
8   imports: [RouterLink],
9   templateUrl: './photos-list.component.html',
10  styleUrls: ['./photos-list.component.scss'],
11 })
12 export class PhotosListComponent {
13   photos = signal<Photo[]>([]);
14   constructor(private photosService: PhotosService) {}
15
16   ngOnInit(): void {
17     this.photosService
18       .getPhotos()
19       .subscribe((res) => this.photos.set(res.slice(0, 20)));
20   }
21 }
22
```

Y ahora modifiquemos la vista para mostrar los nombres de las imágenes y que por cada imagen nos redireccione a la vista detalle pasando por **url** el identificador correcto:

```
photos-list.component.html X
src > app > pages > photos-list > photos-list.component.html > ...
Go to component
1 <div class="container">
2   <h1>Images</h1>
3   <ul class="list-group">
4     @for (img of photos(); track img.id) {
5       <a
6         [routerLink]=["'/photos', img.id]"
7         class="list-group-item list-group-item-action animate__animated animate__bounce animate__fadeIn"
8       >
9         {{ img.id }} - {{ img.author }}
10        </a>
11      }
12   </ul>
13 </div>
14
```

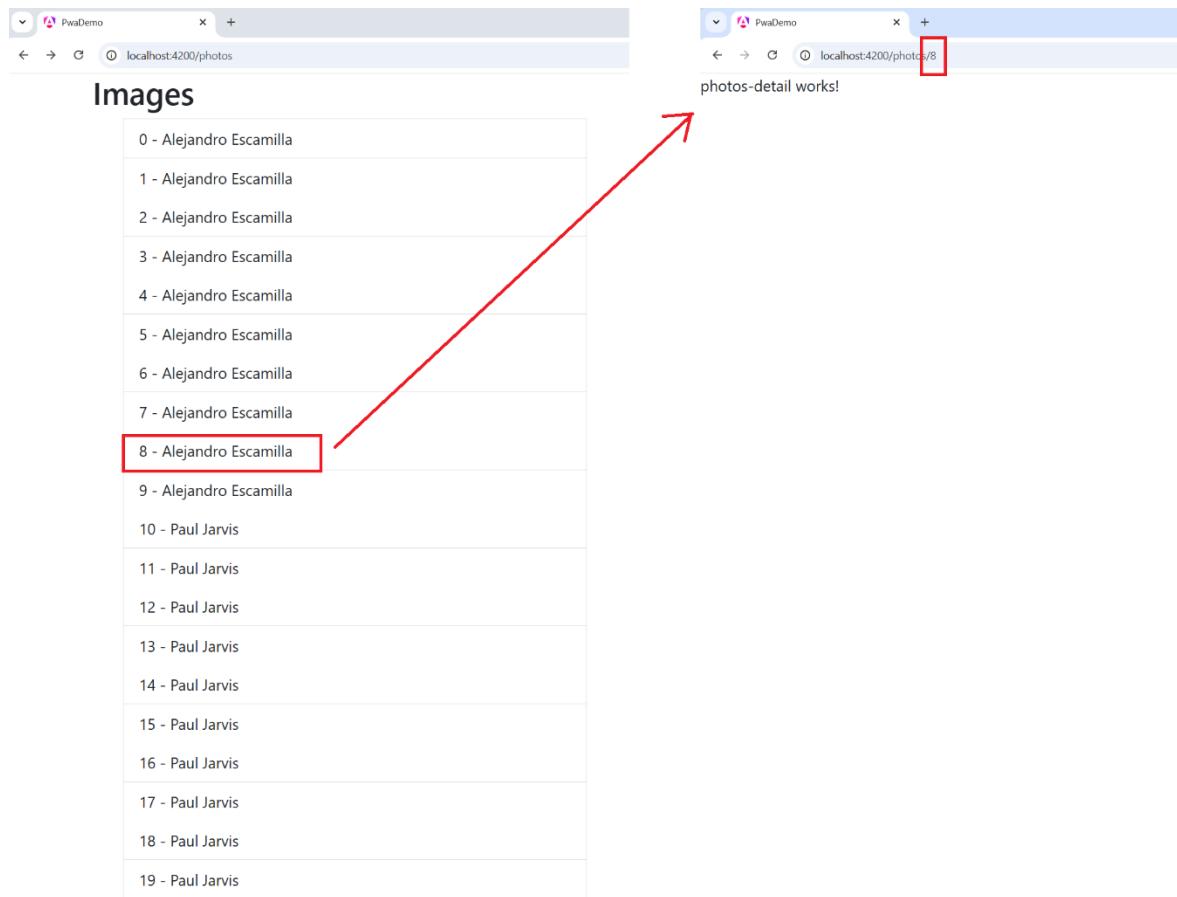
Podemos ver la ejecución:



A screenshot of a web browser window titled "PwaDemo". The address bar shows "localhost:4200/photos". The main content area is titled "Images" and contains a list of 20 items, each consisting of a number followed by a name. The items are:

- 0 - Alejandro Escamilla
- 1 - Alejandro Escamilla
- 2 - Alejandro Escamilla
- 3 - Alejandro Escamilla
- 4 - Alejandro Escamilla
- 5 - Alejandro Escamilla
- 6 - Alejandro Escamilla
- 7 - Alejandro Escamilla
- 8 - Alejandro Escamilla
- 9 - Alejandro Escamilla
- 10 - Paul Jarvis
- 11 - Paul Jarvis
- 12 - Paul Jarvis
- 13 - Paul Jarvis
- 14 - Paul Jarvis
- 15 - Paul Jarvis
- 16 - Paul Jarvis
- 17 - Paul Jarvis
- 18 - Paul Jarvis
- 19 - Paul Jarvis

Y si hacemos clic en cualquier imagen de la lista, podremos ver como iríamos a la vista detalle pasando el identificador correcto por la **url**, por ejemplo:



● Paso 9:

Implementemos la vista detalle **photos-detail.component**, primera versión:

```

src > app > app.routes.ts <-
1 import { Routes } from '@angular/router';
2 import { PhotosListComponent } from './pages/photos-list/photos-list.component';
3 import { PhotosDetailComponent } from './pages/photos-detail/photos-detail.component';
4
5 export const routes: Routes = [
6   { path: '', redirectTo: 'photos', pathMatch: 'full' },
7   { path: 'photos', component: PhotosListComponent },
8   { path: 'photos/:id', component: PhotosDetailComponent },
9 ];
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

```

```

src > app > pages > photos > photos-detail > photos-detail.component.ts <-
1 import { PhotosService } from '../../../../../services/photos.service';
2
3 @Component({
4   selector: 'app-photos-detail',
5   templateUrl: './photos-detail.component.html',
6   styleUrls: ['./photos-detail.component.css'],
7 })
8
9
10 export class PhotoDetailComponent {
11   constructor(
12     private photosService: PhotosService,
13     // Para leer parámetros de la url
14     private activatedRoute: ActivatedRoute,
15     // Para redirigir a una vista determinada si no tenemos un identificador válido
16     private router: Router
17   ) {}
18
19
20   ngOnInit(): void {
21     const identifier = this.activatedRoute.snapshot.paramMap.get('id');
22     console.log('Identifier -->', identifier);
23   }
24 }
25

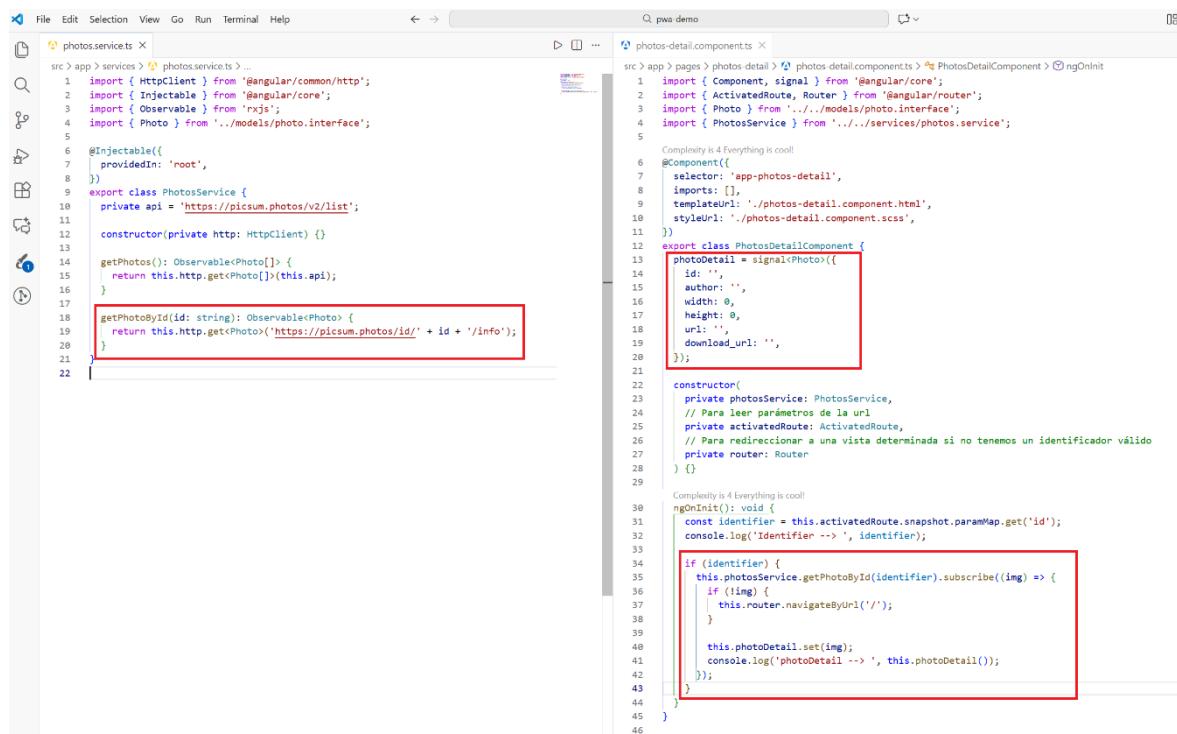
```

En el constructor del **photos-detail.component.ts** he añadido un breve comentario para que se entienda el por qué necesitamos los **imports** de **ActivateRoute** y **Router**.

!!! Tenemos que asegurarnos que el nombre de la variable que pasamos por **url** y que recuperamos en el **photos-detail.component.ts** mediante el **activateRoute** sea el mismo que definimos en el **app.routes.ts**.

Si ejecutamos la aplicación, podremos ver que cada vez que hacemos clic a un elemento de la lista, nos navega a la vista detalle y por consola se muestra el identificador correcto, por lo tanto, somos capaces de capturar el identificador que se pasa por la url correctamente.

Hagamos lo siguiente:



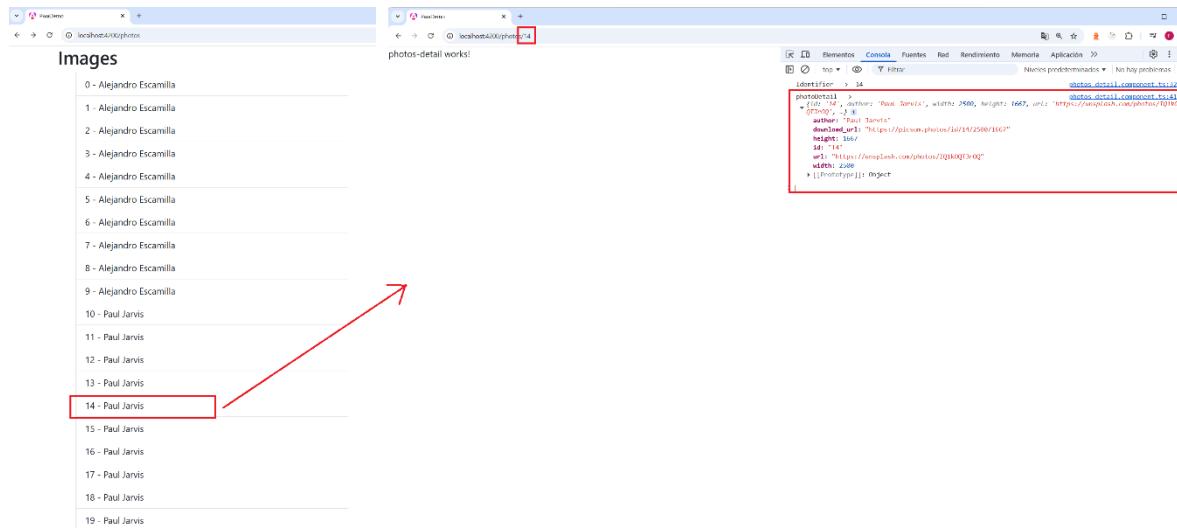
```
photos.service.ts
src > app > services > photos.service.ts > ...
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { Photo } from '../models/photo.interface';
5
6 @Injectable({
7   providedIn: 'root',
8 })
9 export class PhotosService {
10   private api = 'https://picsum.photos/v2/list';
11
12   constructor(private http: HttpClient) {}
13
14   getPhotos(): Observable<Photo[]> {
15     return this.http.get<Photo[]>(this.api);
16   }
17
18   getPhotoById(id: string): Observable<Photo> {
19     return this.http.get<Photo>(`https://picsum.photos/id/${id}/info`);
20   }
21 }

photos-detail.component.ts
src > app > pages > photos > photos-detail.component.ts > PhotosDetailComponent > ngOnInit
1 import { Component, signal } from '@angular/core';
2 import { ActivatedRoute, Router } from '@angular/router';
3 import { Photo } from '../models/photo.interface';
4 import { PhotosService } from '../services/photos.service';
5
6 Complexity is 4 Everything is cool!
7 @Component({
8   selector: 'app-photos-detail',
9   imports: [],
10   templateUrl: './photos-detail.component.html',
11   styleUrls: ['./photos-detail.component.scss'],
12 })
13 export class PhotosDetailComponent {
14   photoDetail = signal<Photo>({
15     id: '',
16     author: '',
17     width: 0,
18     height: 0,
19     url: '',
20     downloadUrl: ''
21   });
22
23   constructor(
24     private photoService: PhotosService,
25     // Para leer parámetros de la url
26     private activatedRoute: ActivatedRoute,
27     // Para redireccionar a una vista determinada si no tenemos un identificador válido
28     private router: Router
29   ) {}
30
31 Complexity is 4 Everything is cool!
32 ngOnInit(): void {
33   const identifier = this.activatedRoute.snapshot.paramMap.get('id');
34   console.log('Identifier -->', identifier);
35
36   if (identifier) {
37     this.photoService.getPhotoById(identifier).subscribe((img) => {
38       if (!img) {
39         this.router.navigate(['/']);
40       }
41
42       this.photoDetail.set(img);
43       console.log('photoDetail -->', this.photoDetail());
44     });
45   }
46 }
```

En el servicio **photos.service.ts** implementemos una función **getPhotoById** en la que pasando un **id** hagamos una llamada a la api y nos devuelva la información concreta de dicha imagen.

Llamemos al controlador **photos-detail.component.ts** este servicio que acabamos de implementar enviando por argumento el identificador que nos viene por **url** y que hemos validado que lo capturamos correctamente.

Vemos la ejecución:



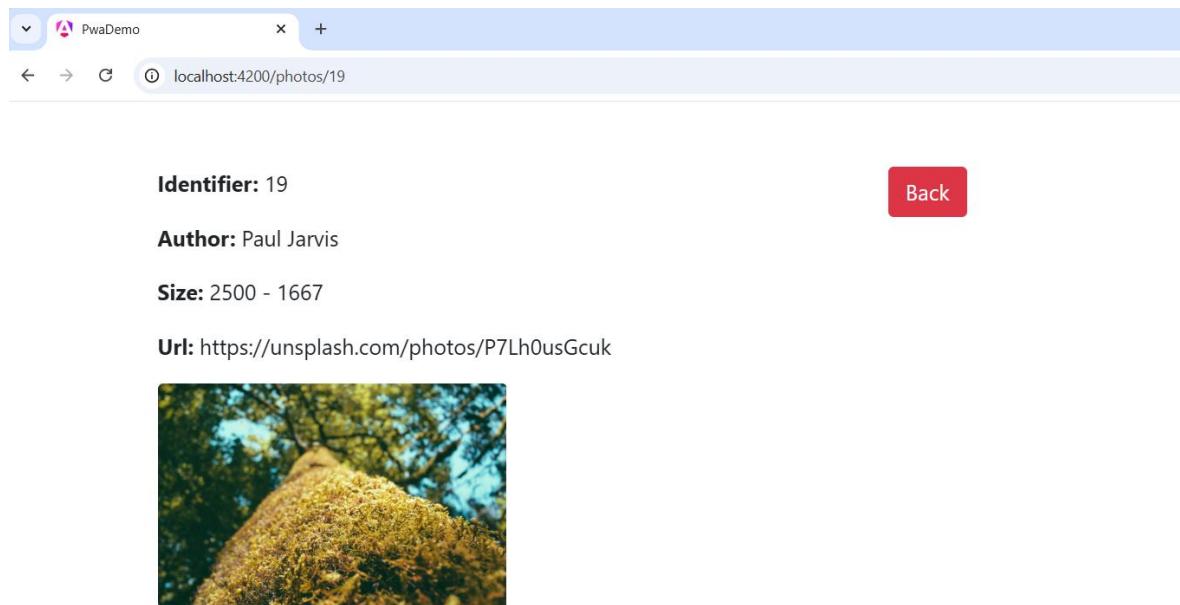
Podemos ver que cuando navegamos a la vista detalle se hace la llamada a la api para recuperar la información de la imagen identificada por el **id** que recuperamos por **url** y mandamos a la api.

Vamos a maquetar la vista detalle:

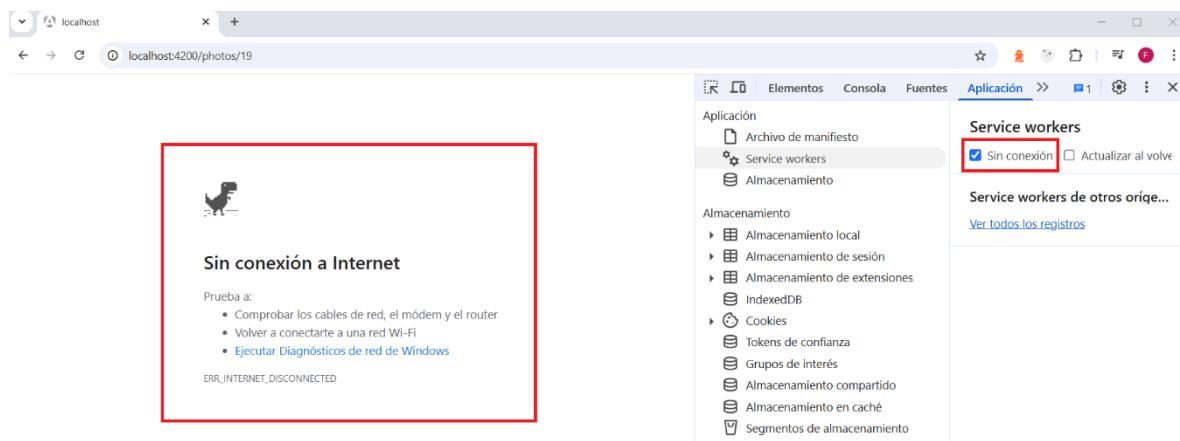
```
src > app > pages > photos-detail > photos-detail.component.html > ...
Go to component
1  <div class="container mt-5">
2    @if(photoDetail()){
3      <div class="row animate__animated animate__bounce animate__fadeIn">
4        <div class="col-md-9">
5          <p><strong>Identifier:</strong> {{ photoDetail().id }}</p>
6          <p><strong>Author:</strong> {{ photoDetail().author }}</p>
7          <p>
8            <strong>Size:</strong> {{ photoDetail().width }} -
9            {{ photoDetail().height }}</p>
10         </p>
11         <p><strong>Url:</strong> {{ photoDetail().url }}</p>
12         
18       </div>
19       <div class="col-md-3">
20         <a class="btn btn-danger" routerLink="/">Back</a>
21       </div>
22     </div>
23   }
24 </div>
```

Vamos a pintar en la vista los campos de la respuesta simplemente para mostrarlos y asegurarnos que funciona bien. Nótese que añadimos un `@if(photoDetail()){` para asegurarnos que mostramos la información cuando ésta esté cargada, evitando así los típicos errores de `undefined`.

Si ejecutamos la aplicación podremos ver que accediendo a cada detalle de cada imagen veremos algo así:



Llegados a este punto, tenemos una aplicación relativamente sencilla implementada en **Angular**. Nótese que, si asignamos el modo **offline** y refrescamos, no podríamos utilizar nuestra aplicación web, en breve veremos cómo podemos manejar esto:



● Paso 10:

Vamos a transformar nuestra aplicación en **PWA**. Como complemento a los pasos que seguiremos a continuación, podemos consultar la documentación oficial en:

<https://v19.angular.dev/ecosystem/service-workers>

El primer paso para transformar nuestra aplicación a **PWA** es ejecutar el siguiente comando:

```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng add @angular/pwa
● ✓ Determining Package Manager
  > Using package manager: npm
✓ Searching for compatible package version
  > Found compatible package version: @angular/pwa@19.2.19.
✓ Loading package information from registry
✓ Confirming installation
✓ Installing package in temporary location
CREATE ngsw-config.json (669 bytes)
CREATE public/manifest.webmanifest (1280 bytes)
CREATE public/icons/icon-128x128.png (2875 bytes)
CREATE public/icons/icon-144x144.png (3077 bytes)
CREATE public/icons/icon-152x152.png (3293 bytes)
CREATE public/icons/icon-192x192.png (4306 bytes)
CREATE public/icons/icon-384x384.png (11028 bytes)
CREATE public/icons/icon-512x512.png (16332 bytes)
CREATE public/icons/icon-72x72.png (1995 bytes)
CREATE public/icons/icon-96x96.png (2404 bytes)
UPDATE angular.json (2820 bytes)
UPDATE package.json (1080 bytes)
UPDATE src/app/app.config.ts (647 bytes)
UPDATE src/index.html (743 bytes)
✓ Packages installed successfully.
```

Fijémonos en los ficheros que nos ha creado en nuestro proyecto.

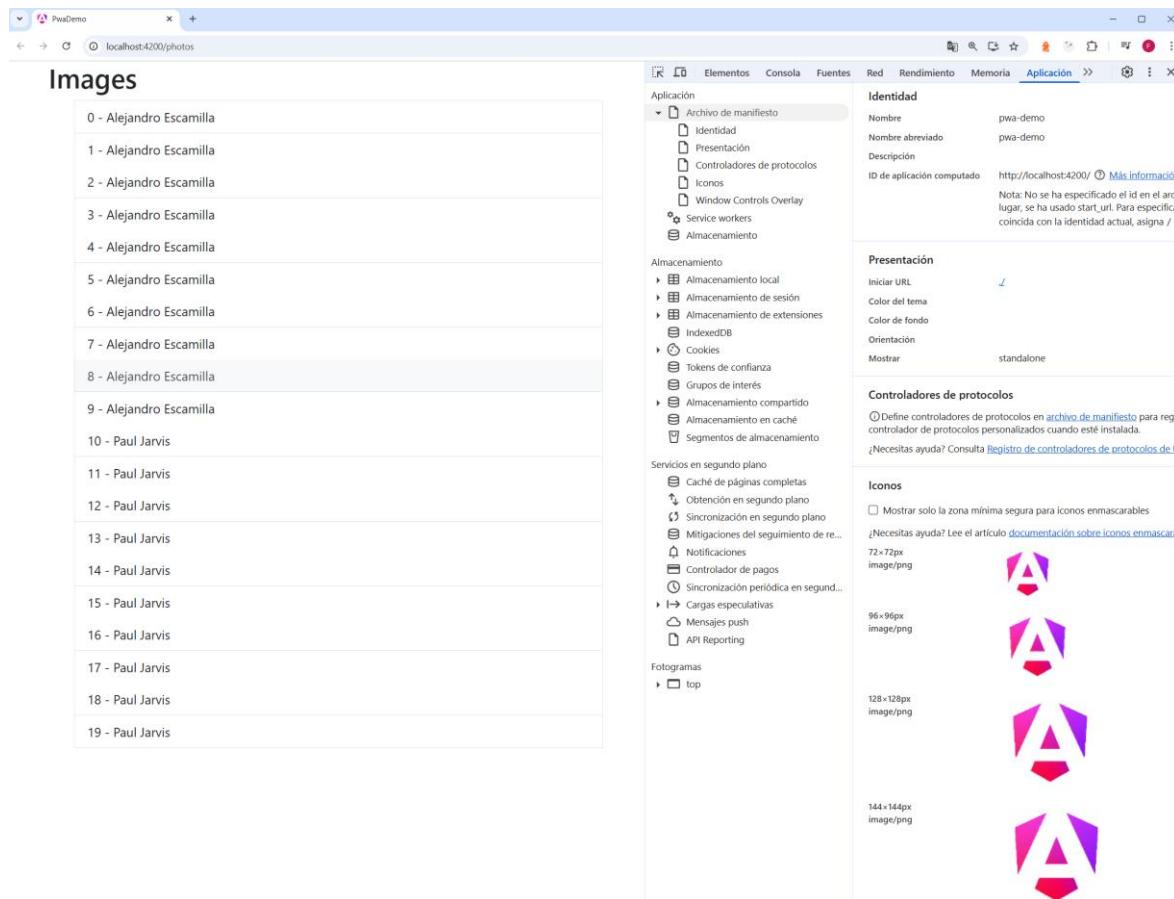
Por una parte, nos ha añadido el **ngsw-config.json** el cual es el **json** en el que haremos todas las configuraciones de lo que será nuestro **service worker**, posteriormente lo veremos.

También podemos fijarnos que nos ha añadido el fichero **manifest.webmanifest** el cual contiene el aspecto visual de nuestra aplicación, posteriormente lo podremos observar en el navegador.

También podemos observar que nos ha creado una carpeta **icons** dentro de **assets** con los diferentes iconos que necesitaremos. Utilizaremos los que trae **Angular** por defecto, en una app real, los sustituiremos por los nuestros.

Finalmente, hay que comentar que nos ha actualizado el fichero **index.html** para vincular el **manifest**.

Si ahora hacemos un **ng serve --open** para ver nuestra aplicación en ejecución, podemos ver que si tenemos activo el **manifest**:



Aquí mostramos los valores por defecto. Esto se podría personalizar, como los colores del tema principal y secundario, los mismos iconos, ...

Pero en cambio no tenemos registrado ningún **service worker**.

Lo que tenemos que hacer es ejecutar la siguiente instrucción:

ng build

Esto lo que nos hará es crear una carpeta **dist** dentro del proyecto y dentro de esta carpeta lo que tendríamos es todo lo que subiríamos a un servidor real para desplegar la aplicación en un entorno de producción.

Es en este 'paquete' donde si nos funcionará el **service worker**.

Claro, ¿aquí nos puede surgir una pregunta, como ejecutamos nuestra aplicación en local como si estuviera desplegado en un servidor real? Recordemos que para poder utilizar una **PWA** necesitamos ejecutar nuestra aplicación con un certificado de seguridad **https**.

Aunque si estamos en **localhost** también nos funcionaría.

La manera más sencilla para probar el **frontend** sería instalar el siguiente paquete:

npm install --global http-server

El cual es un servidor ligero que nos permitiría desplegar nuestra aplicación.

Si vamos a la ruta de donde se nos ha creado nuestro desplegable de **Angular** y ejecutamos **http-server**:

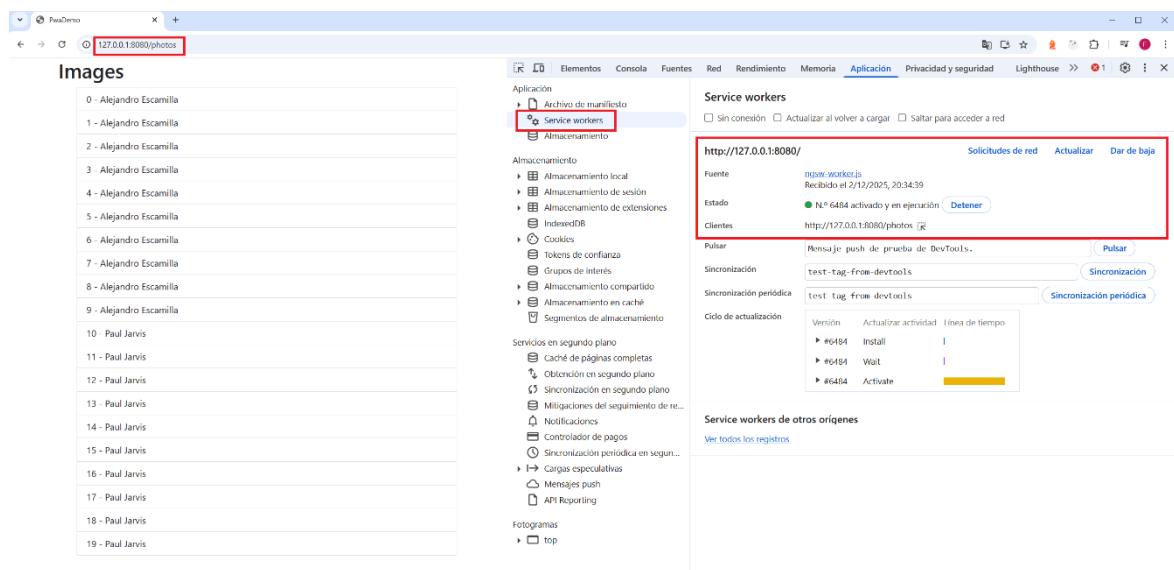
```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo\browser> http-server -p 8080
Starting up http-server, serving .
http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
http://192.168.1.131:8080
http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Esto quiere decir que en <http://127.0.0.1:8080> se está ejecutando nuestra aplicación.

Podemos ver como nuestro **service worker** está registrado y funcionando.



Con este servidor ligero, para hacer pruebas del **frontend** nos valdría.

*No obstante, normalmente necesitaremos montar un **apache** junto con una base de datos local para tener todo el entorno parecido al que podamos tener en el servidor de producción.*

Si ahora ponemos el modo **offline** veremos que la aplicación no funciona correctamente, en breve configuraremos el **service worker** para que podamos trabajar **offline**. Nótese que ahora para poder mostrar la información necesitamos consultar al exterior (internet) los estilos de **Bootstrap**, y las llamadas a los servicios.

Si limpiamos la caché:

The screenshot shows the Chrome DevTools interface with the 'Aplicación' tab selected. In the left sidebar, under 'Almacenamiento', the 'Almacenamiento' section is highlighted with a red box. In the main content area, there is a 'Borrar datos de sitios' button, which is also highlighted with a red box. There is a red arrow pointing from the sidebar highlight to the button highlight.

Y ponemos el modo offline:

The screenshot shows the Chrome DevTools interface with the 'Aplicación' tab selected. In the left sidebar, under 'Service workers', the 'Service workers' section is highlighted with a red box. In the main content area, there is a 'Sin conexión' checkbox, which is also highlighted with a red box.

Resultado:

The screenshot shows a browser window with the URL '127.0.0.1:8080/photos/2'. The console tab is selected, showing an error message: 'Failed to load resource: the server responded with a status of 404 (Not Found)'. Below the error message, there is some detailed developer information about the request.

Veremos que nuestra aplicación sigue funcionando, aunque como podemos ver por la consola, necesita consultar a la api de las imágenes para obtener la información y como no tenemos internet no puede resolver esta información y devuelve error.

Aquí es donde entran las estrategias de la caché que podremos configurar en nuestro **service worker**. Es decir, habrá casos que nos interesaría ir a consultar directamente en internet, pero, en otras ocasiones, nos interesaría consultar primero en la caché, y si no tenemos los datos que nos interese, nos iríamos a pedirlos a internet.

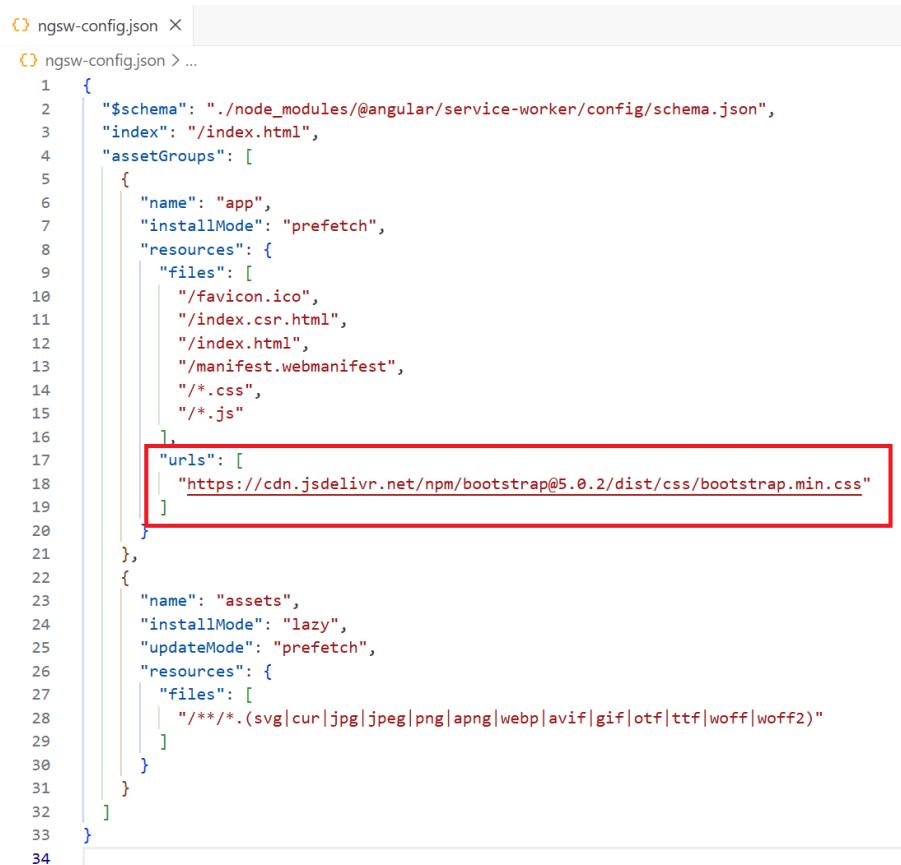
Vamos a hacer las configuraciones necesarias para dejar totalmente funcional nuestra aplicación de ejemplo:

- **Paso 11:**

Configurando nuestro **service worker**:

En el fichero **ngsw-config.json** tenemos:

- **Resources/files:** tenemos lo que sería nuestra **AppShell**, es decir, todos los ficheros indispensables para que funcione nuestra aplicación. Debemos tener en cuenta que todos los recursos de este apartado **files** los busca a partir de la raíz del proyecto.
- ¿Cómo gestionamos la librería de estilos del **Bootstrap** que pusimos dinámicamente en el fichero **index.html**?



```
1  {
2    "$schema": "./node_modules/@angular/service-worker/config/schema.json",
3    "index": "/index.html",
4    "assetGroups": [
5      {
6        "name": "app",
7        "installMode": "prefetch",
8        "resources": {
9          "files": [
10            "/favicon.ico",
11            "/index.csr.html",
12            "/index.html",
13            "/manifest.webmanifest",
14            "/*.css",
15            "/*.js"
16          ],
17          "urls": [
18            "https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
19          ]
20        }
21      },
22      {
23        "name": "assets",
24        "installMode": "lazy",
25        "updateMode": "prefetch",
26        "resources": {
27          "files": [
28            "**/*.(svg|cur|jpg|jpeg|png|apng|webp|avif|gif|otf|ttf|woff|woff2)"
29          ]
30        }
31      }
32    ]
33  }
```

Añadimos la propiedad **urls** y aquí podemos definir las **urls** que necesitamos. Con esto lo que les estaremos diciendo es, para esta **url**, cuando sea requerida, la guardas en caché, para que, de este modo, cuando la volvamos a necesitar, la recoja de caché primero en lugar de ir hacia internet.

Con esto, podemos ver que, por una parte, podríamos utilizar la librería sin internet ya que la recogeremos de la caché, y, por otro lado, todos los recursos que vengan de la caché son servidos más rápido que si vienen de internet con lo que poco a poco, tendremos un mejor rendimiento global de la aplicación.

Nótese que cuando tengamos las configuraciones hechas en nuestro fichero **ngsw-config.json** tendremos que volver a generar el desplegable haciendo **ng build --prod** para ver los cambios aplicados.

Por lo tanto, iríamos a la raíz del proyecto, ejecutaremos la instrucción anterior, luego iríamos a la carpeta **dist** y volveríamos a levantar el servidor para ver los cambios.

```
● > http-server stopped.
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo\browser> cd..
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo> cd..
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist> cd..
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng build
Initial chunk files | Names | Raw size | Estimated transfer size
main-C7DEVS62.js | main | 221.14 kB | 60.08 kB
polyfills-B6TNHZQ6.js | polyfills | 34.58 kB | 11.32 kB
styles-5INURTSO.css | styles | 0 bytes | 0 bytes
| Initial total | 255.72 kB | 71.40 kB

Application bundle generation complete. [1.934 seconds]

▲ [WARNING] Unable to locate stylesheet: C:\assets\css\animate.css

Output location: C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo

● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> cd ..\dist
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist> dir

Directorio: C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist

Mode LastWriteTime Length Name
---- ----- ----- -----
d----- 02/12/2025 20:46 pwa-demo

● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist> cd ..\pwa-demo\
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo> cd ..\browser\
○ PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo\browser> http-server -p 8080
Starting up http-server, serving ..

http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
http://192.168.1.131:8080
http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Si vamos a la url <http://127.0.0.1:8080/> podremos ver cómo podemos ejecutar nuestra aplicación (marcad **update on reload** para que se actualice el **service-worker**) y si luego ponemos el modo **offline** podremos ver como igualmente tenemos los estilos de **bootstrap** ya que los consulta des de la caché.

La caché la podemos consultar en la misma pestaña **Application** en el apartado **Cache**.

Vamos ahora a manejar las llamadas a la api para poder trabajar sin conexión:

```
① ngrc ngsn-config.json X
② ngrc ngsn-config.json > ...
1  {
2   "$schema": "./node_modules/@angular/service-worker/config/schema.json",
3   "index": "/index.html",
4   "assetGroups": [
5     {
6       "name": "app",
7       "installMode": "prefetch",
8       "resources": {
9         "files": [
10           "/favicon.ico",
11           "/index.csr.html",
12           "/index.html",
13           "/manifest.webmanifest",
14           "/*.css",
15           "/*.js"
16         ],
17         "urls": [
18           "https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
19         ]
20       }
21     },
22     {
23       "name": "assets",
24       "installMode": "lazy",
25       "updateMode": "prefetch",
26       "resources": {
27         "files": [
28           "**/*.(svg|cur|jpg|jpeg|png|apng|webp|avif|gif|otf|ttf|woff|woff2)"
29         ]
30       }
31     }
32   ],
33   "dataGroups": [
34     {
35       "name": "photos-api",
36       "urls": [https://picsum.photos/v2/list],
37       "cacheConfig": {
38         "maxSize": 100,
39         "maxAge": "1h",
40         "timeout": "2s",
41         "strategy": "freshness"
42       }
43     }
44   ]
45 }
```

En este bloque **dataGroups** gestionaremos las estrategias de las llamadas a las apis:

Le pondremos por ejemplo el nombre **photos-api**.

En **urls** de momento le ponemos la **url** de la llamada que nos devuelve el listado de imágenes.

Y luego en **cacheConfig** le decimos:

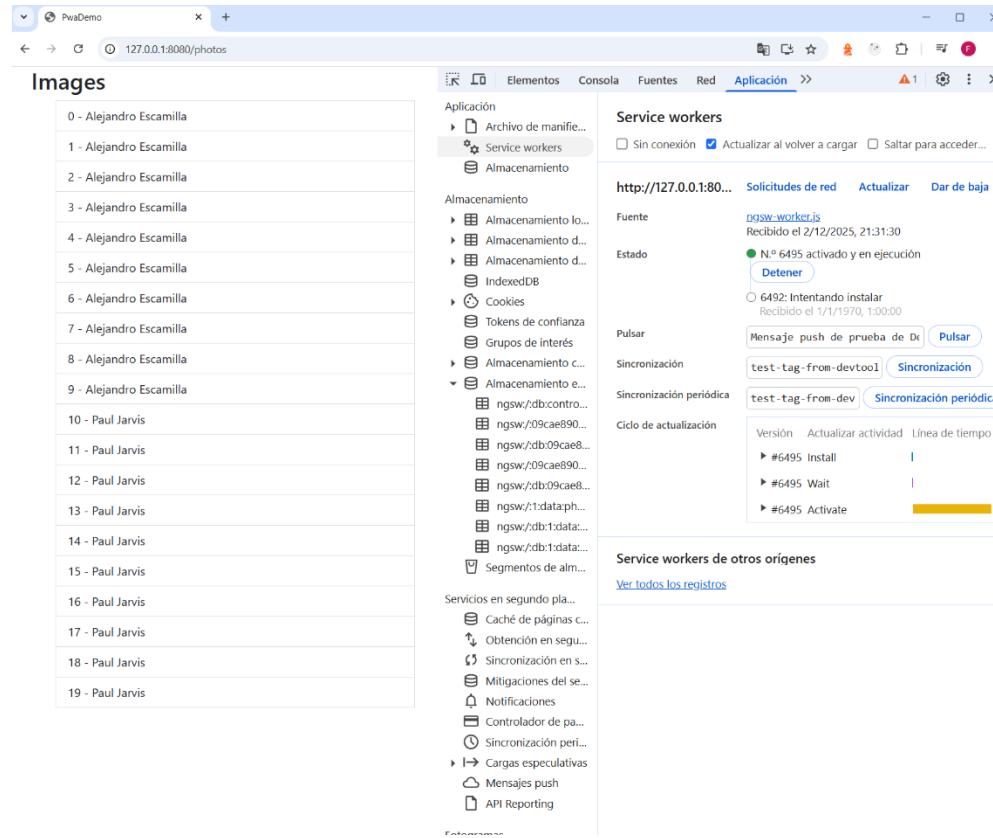
- **maxSize** es el número máximo de respuestas que se guardarán en caché, por ejemplo, le ponemos 100.
- **maxAge** es cuánto tiempo guardaremos en caché dicha información, por ejemplo, le ponemos una hora.
- **timeout** es cuánto tiempo esperas a tener respuesta, si no tenemos respuesta de internet nos vamos a buscar la información en la caché, por ejemplo, le ponemos dos segundos.
- **strategy** es la estrategia de caché que queremos utilizar, por ejemplo, le ponemos **freshness** con lo que le estamos diciendo, primero internet, luego caché, por el hecho de que en este caso me interesa tener los últimos datos actualizados, y si por lo que sea no tuviera internet, al menos, podría seguir utilizando la aplicación con los datos de la caché, aunque estos no fueran los últimos.

Este es un simple caso de ejemplo, consulta la documentación oficial para ver las diferentes posibilidades de estrategias y configuraciones para adaptarlas correctamente a vuestros proyectos.

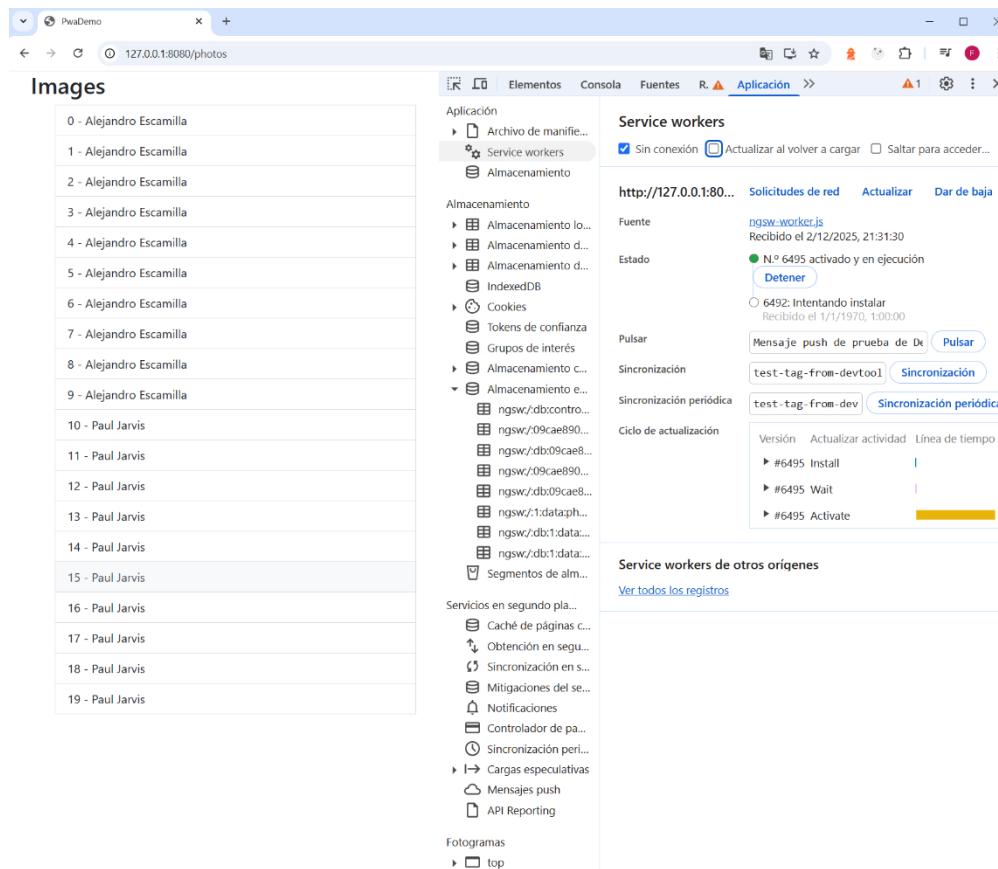
Vamos a probar estos cambios.

Recordemos, hacemos el **build**, volvemos a levantar la aplicación con el servidor **http-server**, sin estar marcado **offline** y estando marcado **update on reload** actualizamos el navegador, y luego marcamos el modo **offline** sin estar marcado **update on reload**.

Es decir, primero;

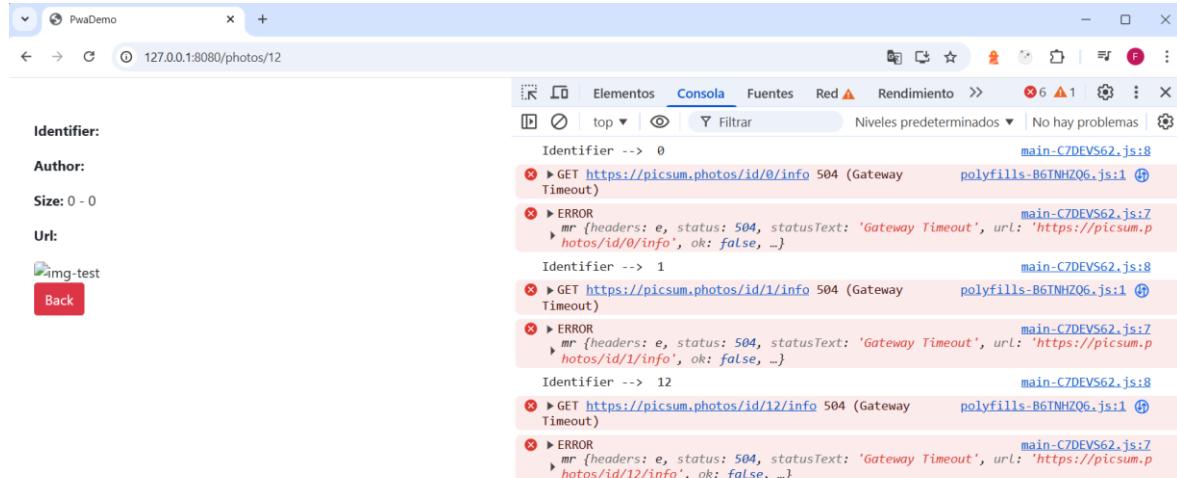


Luego:



En este segundo caso, al refrescar el navegador, podremos ver que igualmente nos carga el listado de imágenes, y esto es debido a que, al no tener internet, lo consulta de la caché. No tendríamos los últimos datos en una app real, pero al menos, tendríamos la app funcional.

¿Vale, pero qué pasa si vamos a una vista detalle estando offline?



Pues que nos fallará, ya que esta url no la hemos gestionado.

Aquí lo que podemos hacer es modificar la url de esta manera:

```
"dataGroups": [ { "name": "images-api", "url": [ "https://picsum.photos/**" ] }, { "cacheConfig": { "maxSize": 10, "maxAge": "1h", "timeout": "1s", "strategy": "freshness" } } ]
```

De esta manera le decimos que trate con la estrategia que nosotros le definimos todo lo que proceda de:

<https://picsum.photos/>

Que en nuestro caso nos vale tanto para la llamada que devuelve el listado de imágenes como la llamada que nos devuelve la información de una imagen concreta y además nos guardaría en caché las propias imágenes.

Compilamos el proyecto y volvamos a ejecutarlo.

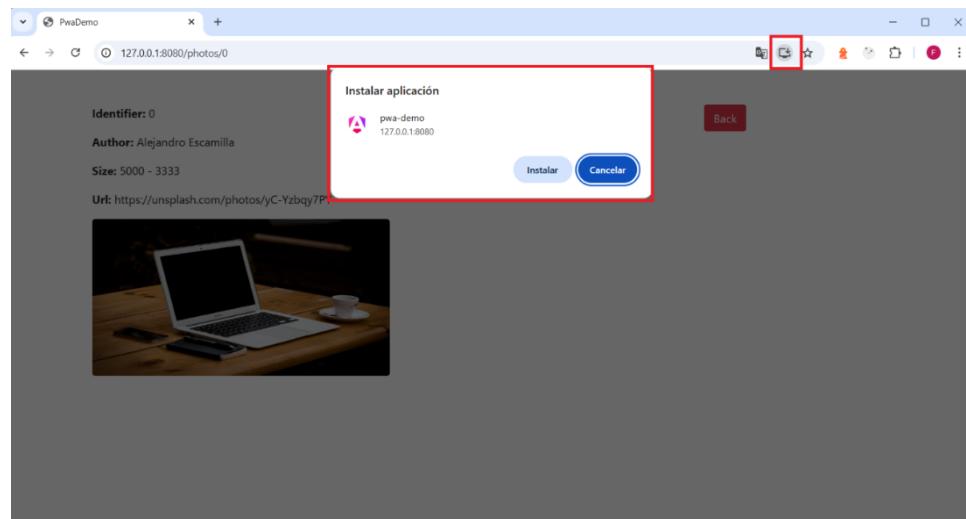
Recordar en limpiar caché, ejecutar con **update on reload** marcado, luego vamos al detalle por ejemplo de las dos primeras imágenes, y luego marcamos el modo **offline**.

Podremos ver que la página del listado de imágenes funciona correctamente, y que podremos acceder al detalle de las dos primeras imágenes ya que hemos accedido previamente a ellas y aunque ahora no tengamos internet, por nuestra estrategia definida, las recupera de la caché.

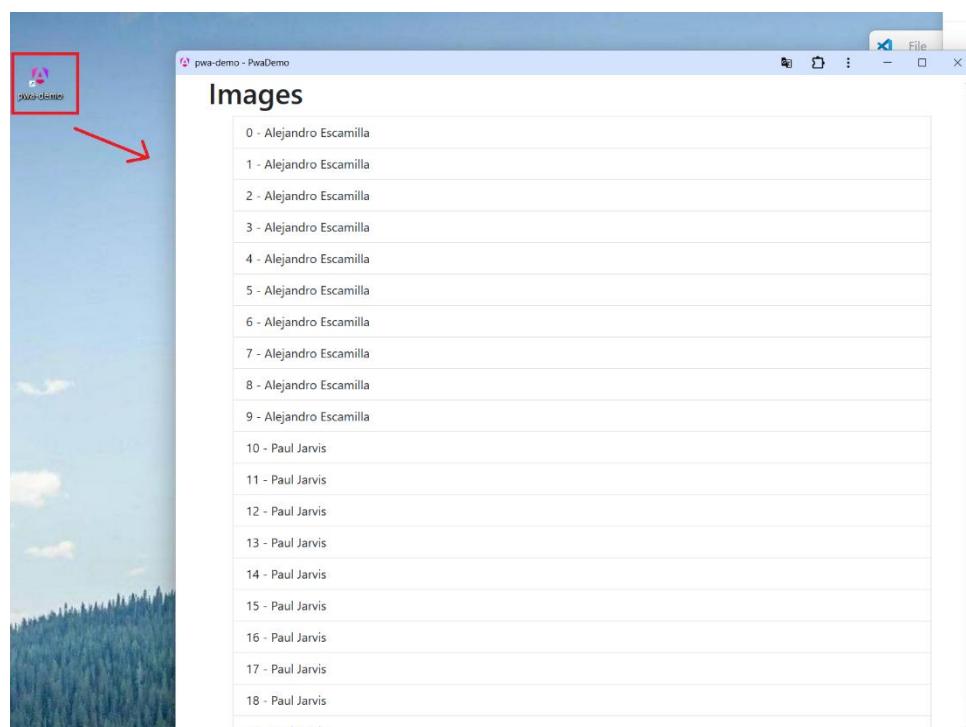
Hay que tener en cuenta que el resto de las imágenes no podríamos acceder a su detalle ya que necesitamos al menos haber accedido una vez con internet para guárdalas en caché.

¿Y si queremos instalar nuestra app?

Si estamos en un ordenador, se habilita un botón para poder instalar nuestra aplicación:



Si instalamos la app:



Podemos ver el icono en el escritorio, y si lo ejecutamos se nos abrirá nuestra app.

El icono, el nombre del icono, el nombre de la app, el color de la parte superior de la app, ... todo esto viene definido del fichero **manifest**.

La gracia de esto es que si accedemos desde un teléfono móvil a una **url** que es una **pwa**, al cabo de unos segundos de acceder nos aparecerá un mensaje que nos dirá si queremos instalar dicha **app**. Si le decimos que sí, se instalará la **app** en un momento y tendremos el icono en el escritorio del móvil como si de otra **app** nativa se tratara.

Esto es muy interesante porque en vez de realizar una **app** nativa para **Android** y una **app** nativa para **iOS**, puedes hacer un único desarrollo web y luego transformarlo a **PWA** con lo que cubres todas las plataformas importantes.

Si que es verdad que en **iOS** hay alguna limitación, como por ejemplo la gestión de las notificaciones **push** que da algunos problemas, pero en principio son cosas que se irán resolviendo.

Una vez estudiada toda esta parte, sería interesante desplegar esta aplicación en algún repositorio público tipo **github** para que vieraís el comportamiento en el móvil.

5. Despliegue

Vamos a desplegar la aplicación en Github así podremos ver nuestras aplicaciones directamente instaladas en nuestros teléfonos.

Requisitos previos:

- Necesitaremos tener instalado Git en nuestro ordenador
 - <https://git-scm.com/install/windows>
- Tener una cuenta creada de Github y un repositorio creado
 - <https://github.com/>

La idea es que os creáis una cuenta. Una vez creada, realizaremos los siguientes pasos:

- Subir el proyecto al repositorio del github
- Una vez subido, lo único que necesitamos hacer es desplegar la aplicación, los pasos son:
 - Ejecutamos las siguientes instrucciones para instalar los paquetes necesarios:

```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> npm i angular-cli-ghpages
added 44 packages, and audited 995 packages in 7s
168 packages are looking for funding
  run `npm fund` for details
2 critical severity vulnerabilities

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng add angular-cli-ghpages
Skipping installation: Package already installed
UPDATE angular.json (2904 bytes)
```

- Luego, tendremos que configurar git y podremos desplegar la aplicación:

```
ng deploy --base-href=<repositoryname>/
```

En mi caso el repositorio se llama “testdeploy”

Os muestra mi salida:

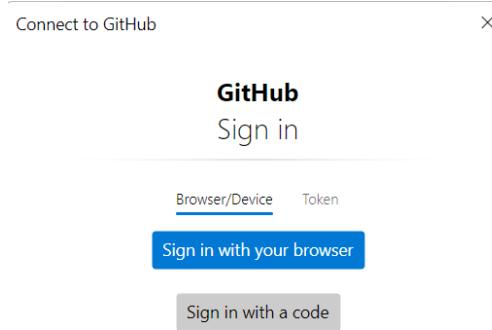
```
PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> git init
Initialized empty Git repository in C:/Users/falbu/Desktop/UOC/2024-2025/2025-1/TEMA4-NUEVO-2025/pwa-demo/.git/
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> git remote add origin https://github.com/falbuera/testdeploy.git
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> git remote -v
origin https://github.com/falbuera/testdeploy.git (fetch)
origin https://github.com/falbuera/testdeploy.git (push)
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> git config --global user.name "Francesc Albuera"
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> git config --global user.email "falbuera@uoc.edu"
● PS C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo> ng deploy --base-href=/testdeploy/
✖ Building "pwa-demo"
✖ Build target "pwa-demo:build:production"
Initial chunk files:
  Names           Raw size | Estimated transfer size
main-C7DEV562.js | main      | 221.14 kB | 60.08 kB
polyfills-BGTHMZQ6.js | polyfills | 34.58 kB | 11.32 kB
styles-SINURTSO.css | styles    | 0 bytes   | 0 bytes
  | Initial total | 255.72 kB | 71.40 kB

Application bundle generation complete. [1.948 seconds]
▲ [WARNING] Unable to locate stylesheet: C:\assets\css\animate.css

Output location: C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\dist\pwa-demo

404.html file created
.nojekyll file created
⚡ Uploading via git, please wait...
Cloning https://github.com/falbuera/testdeploy.git into C:\Users\falbu\Desktop\UOC\2024-2025\2025-1\TEMA4-NUEVO-2025\pwa-demo\node_modules\.cache\gh-pages\https\github.com\falbuera\testdeploy.git
Fetching
Cleaning
Fetching origin
Checking out origin/gh-pages
Removing files
Copying files
Adding all
Committing
Pushing
Info: please complete authentication in your browser...
🌟 Successfully published via angular-cli-ghpages! Have a nice day!
```

Si es la primera vez que utilizamos Git, es posible que nos aparezca una ventana de autenticación:



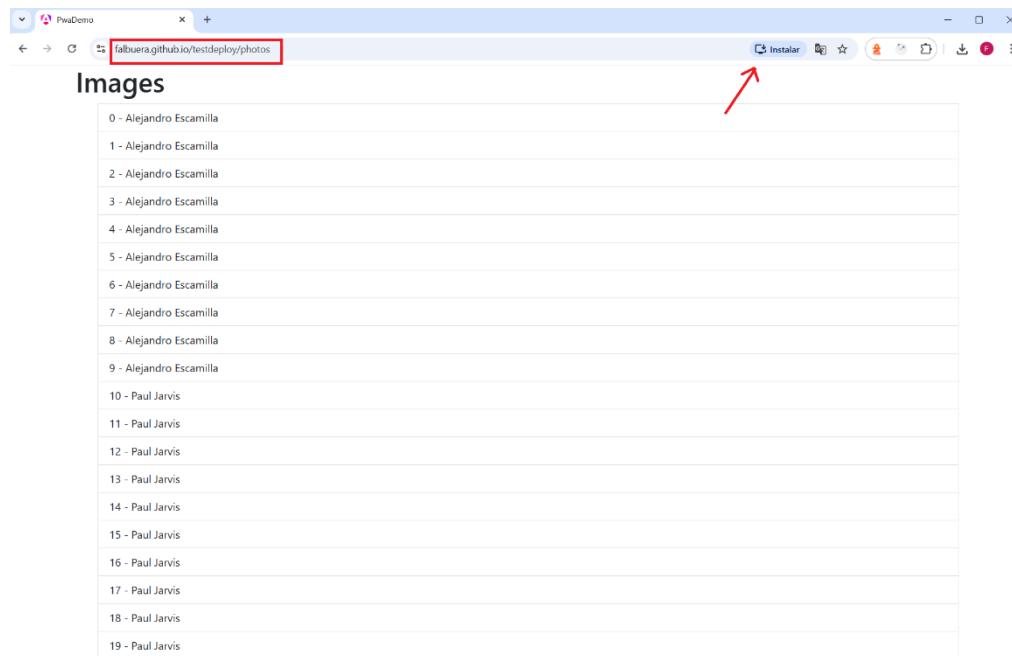
En mi caso utilicé la opción de “Sign in with your browser”. Al estar autenticado en el navegador, la autenticación es rápida, solo es aceptar la siguiente pantalla que aparece.

Finalmente, si vamos al github > Settings > Pages:

The screenshot shows the GitHub Pages settings page for a repository named 'falbuera / testdeploy'. The 'Pages' tab is selected in the left sidebar. A red box highlights the 'Settings' tab in the top navigation bar and the 'GitHub Pages' section in the main content area. The 'GitHub Pages' section displays the message: 'Your site is live at <https://falbuera.github.io/testdeploy/>' and 'Last deployed by falbuera 3 minutes ago'. It also shows the build configuration: 'Deploy from a branch' set to 'gh-pages' and 'Build directory' set to '/ (root)'. Below this, there are sections for 'Custom domain', 'Enforce HTTPS', and 'Visibility'.

Veremos la url que se ha generado y des de la cual podemos acceder a nuestra aplicación desplegada. También podemos observar el estado del deploy.

Si accedemos a la url:



Podemos ver la aplicación funcionando.

Evidentemente tenemos limitaciones, ya que esto es un frontend que no consume ningún backend. No tenemos backend ni base de datos. No obstante, para hacer desarrollo frontend, prototipos, o simplemente desarrollos con fines educativos nos puede venir bien.

En este punto es interesante acceder desde vuestro teléfono e instalar vuestra aplicación.

Observad que “parece” una app nativa una vez instalada.