Open in app Get started

Published in NotOnlyCSS

Luca Spezzano Follow
Oct 8, 2019 · 7 min read ★ · Listen

Save

Turn Your Website into a PWA

Discover how to make a progressive web app in 5 mins and why you need it

Photo by Yura Fresh on Unsplash

According to this article by **CNBC**, Nearly three-quarters of the world will use just their **smartphones** to access the internet **by 2025**.

Starting from a PWA could be the perfect start for your projects, you don't need too much effort and you can build one application that works for all platforms!

What is a PWA?

A Progressive Web App (**PWA**) is a **hybrid** of a regular **web page** and a **mobile application**.
A **PWA** combines features offered by most modern browsers with the benefits of the mobile experience. They are built using standard web technologies, including HTML, CSS, and JavaScript.
The functionalities include **working offline**, **push notifications**, and **device hardware access** and enabling creating user experiences **similar** to **native applications.**

> You can add them to the home of your phone exactly as a mobile app

Why should you build a PWA?

A few weeks ago, I was looking for more information about **PWA,** and I found a very cool website pwastats.com, it shows the stats of all the most famous companies which are using **PWA**.
It was impressive to find out how the companies improved the performance of their website, mostly more than 100%.

A Progressive Web App can **cut load times**, allow users to **navigate offline**, **increase the time spent** on the website, **increase revenue**, can be **much smaller** than a mobile app, and much more.

Who is already using PWA?

Uber, Instagram, Twitter, Pinterest, Forbes, Alibaba

It's interesting to see the companies which already have a **PWA**, some of the world's biggest company such as **Twitter**, **Instagram**, **Uber**, **Pinterest**, **Forbes**, **Alibaba** and much more.

How to make a PWA

To make a **Progressive Web App** at first, we need to develop a responsive website. Once you have done this, we need only two things: a **manifest.json** and a **service worker**. Let's see what they are!

The Web App Manifest

The web app manifest is a simple JSON file that tells the browser about your web application and how it should behave when added on the user's mobile device or desktop.

The properties:

- **name:** is the name used in the app install prompt
- **short_name:** is the name used on the user's home screen, launcher, or other places where space may be limited

feel like a standalone native app.

- **background_color:** is used on the splash screen when the application is launched
- **theme_color:** it sets the color of the toolbar
- **orientation:** it allows to enforce a specific orientation
- **scope:** to be honest usually I don't use this property, but it defines the set of URLs that the browser considers being within your app and often it's used to decide when the user has left the app
- **icons:** when a user adds your site to their home screen, you can define a set of images for the browser to use

The Service Worker

It's essentially a JavaScript file that runs separately from the main browser thread, with a Service Worker you can:

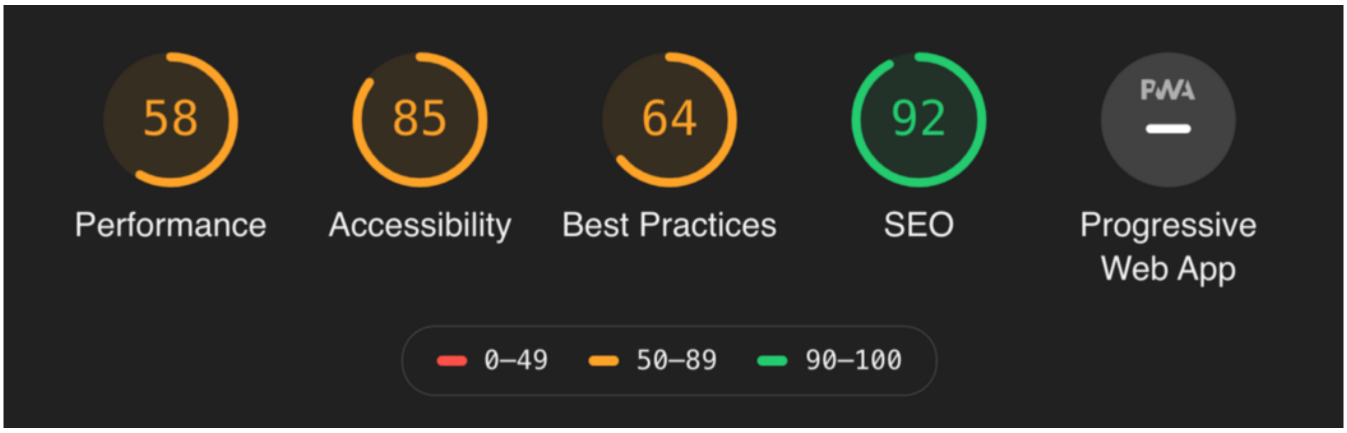
- **intercepting network requests**
- **caching or retrieving resources from the cache**
- **delivering push messages**

Service worker lifecycle

- **Registration:** tells the browser where your service worker is located, and to start installing it in the background.
- **Installation:** triggers an install event where we can run some tasks when the service worker installs.
- **Activation:** if there are any open pages controlled by the previous service worker, the new service worker enters a waiting state. The new service worker only activates when there are no longer any pages loaded that are still using the old service worker and can be used to run some tasks too.

Let's build our PWA step by step in 5 minutes

Before to start I suggest you install the **Lighthouse** extension.
Lighthouse is a tool (by **Google**) for improving the quality of web pages, and it gives you a report like this

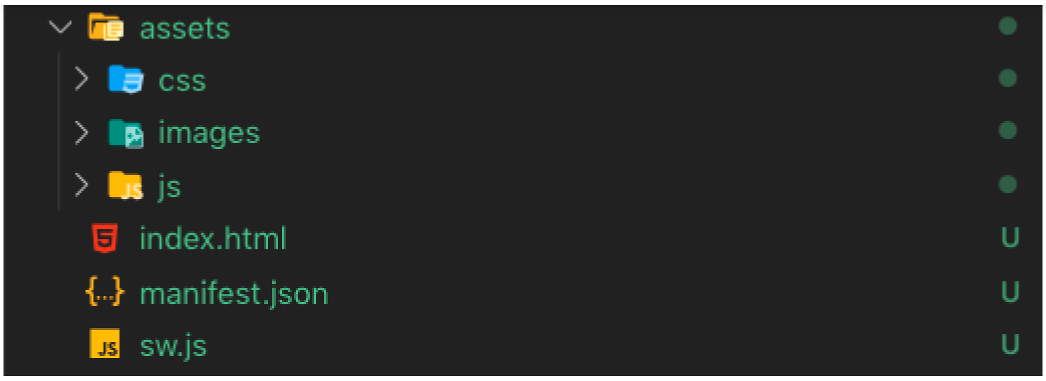


Lighthouse report

and from there, you can check which issues you need to solve to improve **performance**, **accessibility**, **best practices**, **SEO** and **PWA** of your website or web app.

You can install the extension for Chrome here.

The file structure in our example will look like this



```
assets
  css
  images
  js
index.html      U
manifest.json   U
sw.js           U
```

file structure

You will find the full code on Github here, and you can switch branches to see the dynamic or the static cache.

In our `index.html` let's call our `manifest.json`

```
<link rel="manifest" href="/manifest.json">
```

But we also need to call our `app.js` file(where we will register our service worker) and some meta tag necessary to optimize our **PWA**

```
<link rel="apple-touch-icon" href="/assets/images/logo-96x96.png">
<meta name="apple-mobile-web-app-status-bar" content="#FFE1C4">
<meta name="theme-color" content="#FFE1C4">
<script src="/assets/js/app.js"></script>
```

these are the main tags, but of course, we will have more, and they can also have different paths, it is up to you!

Cache static

To begin, we are going to implement a static cache, that's mean that manually we are going to say precisely which resources to put in our cache, for example, all our images, CSS and js files.

This method is useful, especially when you want to download all the resources of your

```
{
  "name": "Name Website",
  "short_name": "NameWebsite",
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#F4F4F4",
  "theme_color": "#F4F4F4",
  "orientation": "portrait-primary",
  "icons": [
    {
      "src": "/assets/images/logo-72x72.png",
      "type": "image/png",
      "sizes": "72x72"
    },
    {
      "src": "/assets/images/logo-96x96.png",
      "type": "image/png",
      "sizes": "96x96"
    },
    {
      "src": "/assets/images/logo-128x128.png",
      "type": "image/png",
      "sizes": "128x128"
    },
    {
      "src": "/assets/images/logo-144x144.png",
      "type": "image/png",
      "sizes": "144x144"
    },
    {
      "src": "/assets/images/logo-152x152.png",
      "type": "image/png",
      "sizes": "152x152"
    },
    {
      "src": "/assets/images/logo-192x192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/assets/images/logo-384x384.png",
      "type": "image/png",
      "sizes": "384x384"
    },
    {
      "src": "/assets/images/logo-512x512.png",
      "type": "image/png",
      "sizes": "512x512"
```

After we need to check if the browser allows service workers, and if so, we will register our service worker in our `app.js` .

```
if('serviceWorker' in navigator){
  navigator.serviceWorker.register('/sw.js')
    .then(reg => console.log('service worker registered'))
    .catch(err => console.log('service worker not registered',
err));
}
```

Now let's write our service worker in our `sw.js` file

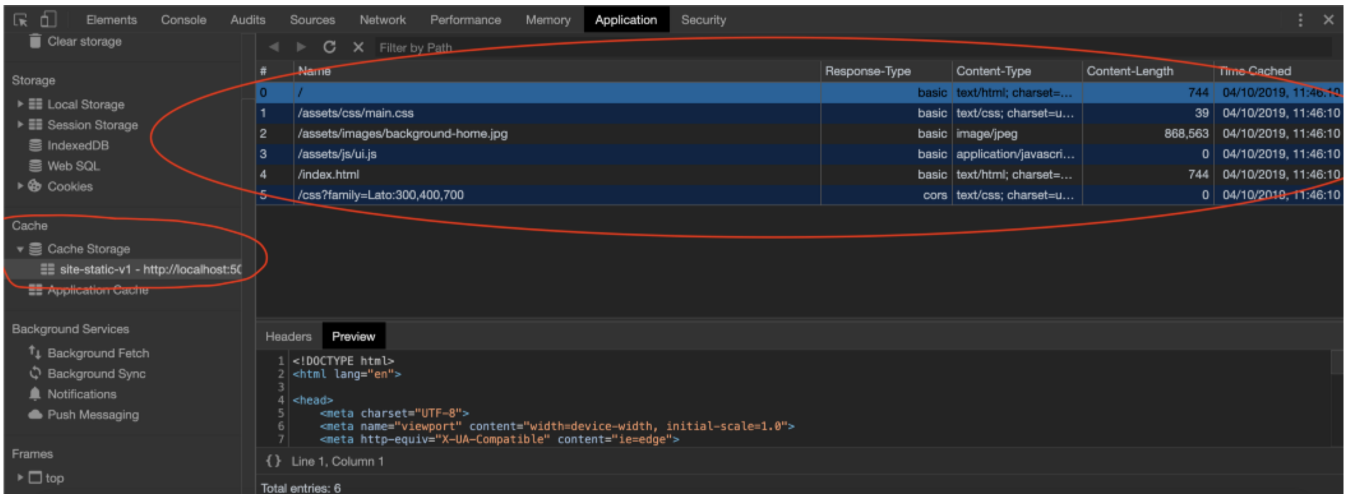
```
const staticCacheName = 'site-static-v1';
const assets = [
  '/',
  '/index.html',
  '/assets/js/ui.js',
  '/assets/css/main.css',
  '/assets/images/background-home.jpg',
  'https://fonts.googleapis.com/css?family=Lato:300,400,700',
];

// install event
self.addEventListener('install', evt => {
  evt.waitUntil(
    caches.open(staticCacheName).then((cache) => {
      console.log('caching shell assets');
      cache.addAll(assets);
    })
  );
});

// activate event
self.addEventListener('activate', evt => {
  evt.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(keys
        .filter(key => key !== staticCacheName)
        .map(key => caches.delete(key))
      );
    })
  );
});
```

```
          return cacheRes || fetch(evt.request);
        })
      );
    });
```

We store in the array all the resources that we want to put in our cache

Install event

we add to the cache all the static assets, and we can see it in our Chrome Console



#	Name	Response-Type	Content-Type	Content-Length	Time Cached
0	/	basic	text/html; charset=…	744	04/10/2019, 11:46:10
1	/assets/css/main.css	basic	text/css; charset=u…	39	04/10/2019, 11:46:10
2	/assets/images/background-home.jpg	basic	image/jpeg	868,563	04/10/2019, 11:46:10
3	/assets/js/ui.js	basic	application/javascri…	0	04/10/2019, 11:46:10
4	/index.html	basic	text/html; charset=…	744	04/10/2019, 11:46:10
5	/css?family=Lato:300,400,700	cors	text/css; charset=u…	0	04/10/2019, 11:46:10

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

Line 1, Column 1

Total entries: 6

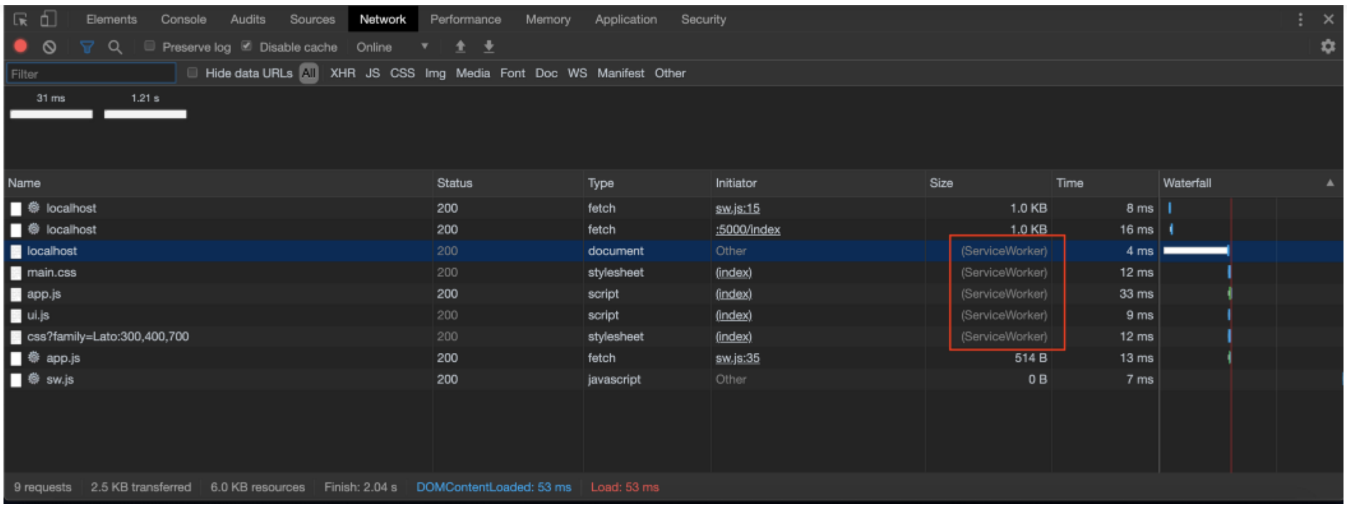
cache screen

Activate event

When we change the name of our cache, we could have multiple caches stored, and that could create problems. To avoid that, we need to delete the old one. In this function we check the key (our cache name), if the same is different from the previous, we delete the previous. Doing that, we always have only the last cache.

Fetch event

Here we check if already exists a cache if it exists we don't download the resources anymore, but we take them from the cache, and we can see it again in the console.



Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	fetch	sw.js:15	1.0 KB	8 ms	
localhost	200	fetch	:5000/index	1.0 KB	16 ms	
localhost	200	document	Other	(ServiceWorker)	4 ms	
main.css	200	stylesheet	(index)	(ServiceWorker)	12 ms	
app.js	200	script	(index)	(ServiceWorker)	33 ms	
ui.js	200	script	(index)	(ServiceWorker)	9 ms	
css?family=Lato:300,400,700	200	stylesheet	(index)	(ServiceWorker)	12 ms	
app.js	200	fetch	sw.js:35	514 B	13 ms	
sw.js	200	javascript	Other	0 B	7 ms	

9 requests | 2.5 KB transferred | 6.0 KB resources | Finish: 2.04 s | DOMContentLoaded: 53 ms | Load: 53 ms

network screen

ps. When we change our service worker file, we need to change the name of the cache. This allows us to update our service worker and create a new cache.

Dynamic cache

The dynamic cache is very powerful because it will cache all the fetch requests **automatically** during your navigation. This cache must be used carefully because, if you use it when you have an API call, you wouldn't see the new data when they change.

All the configuration that you have seen before it's perfectly good you need to change your `sw.js` file with this:

```
const dynamicCacheName = 'site-dynamic-v1';

// activate event
self.addEventListener('activate', evt => {
  evt.waitUntil(
    caches.keys().then(keys => {
      return Promise.all(keys
        .filter(key =>  key !== dynamicCacheName)
        .map(key => caches.delete(key))
      );
    })
  );
});
```

```
            cache.put(evt.request.url, fetchRes.clone());
            return fetchRes;
          })
        });
      })
    );
  });
```

Active event

Here we do the same thing of the static cache

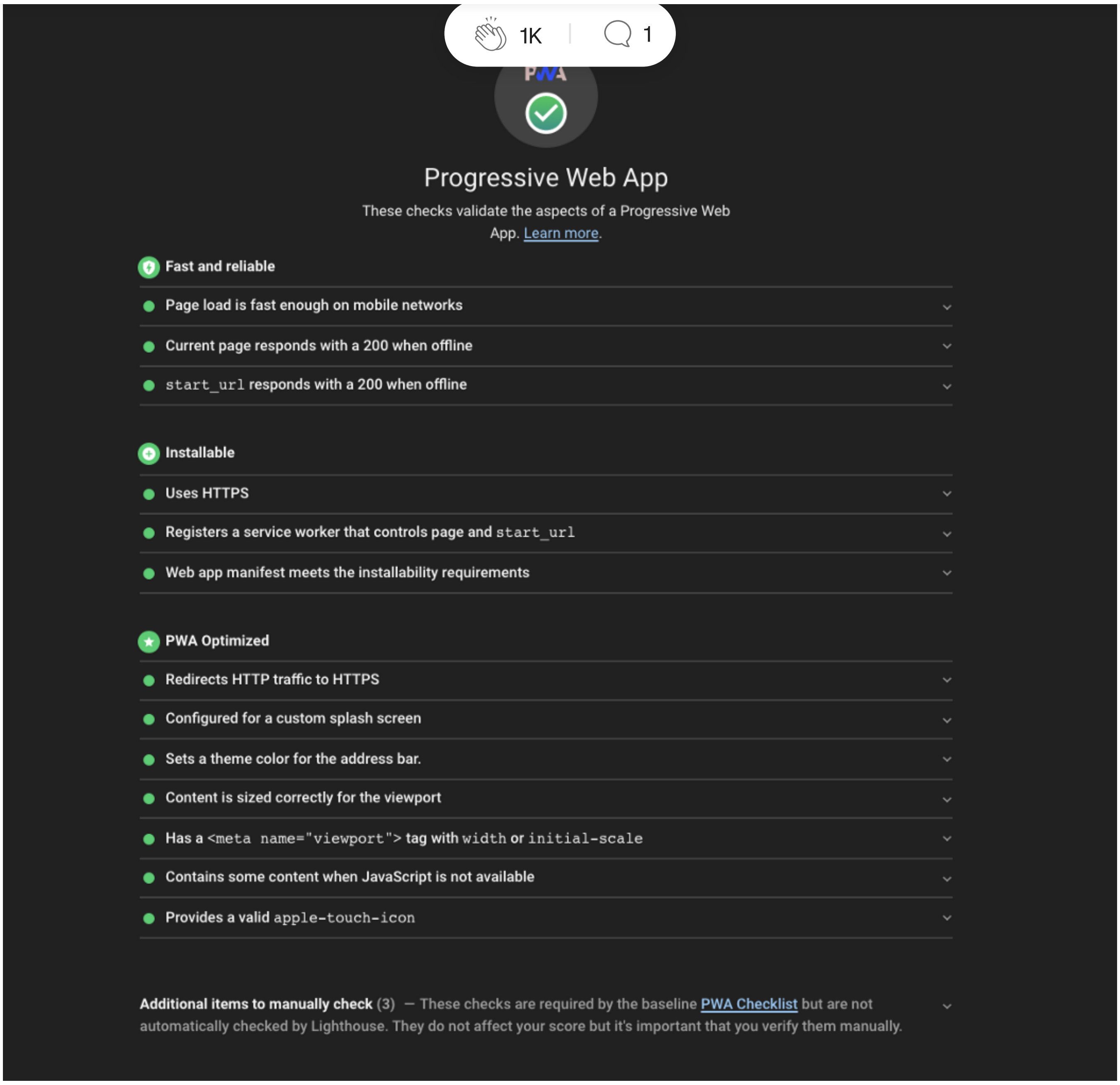
Fetch event

In the fetch event, we put in the cache automatically all the fetch requests.

If you try both the cache and after you run Lighthouse, you can see that your website is a WPA.

Open in app Get started

1K 1



Progressive Web App

These checks validate the aspects of a Progressive Web App. Learn more.

Fast and reliable

- Page load is fast enough on mobile networks
- Current page responds with a 200 when offline
- `start_url` responds with a 200 when offline

Installable

- Uses HTTPS
- Registers a service worker that controls page and `start_url`
- Web app manifest meets the installability requirements

PWA Optimized

- Redirects HTTP traffic to HTTPS
- Configured for a custom splash screen
- Sets a theme color for the address bar.
- Content is sized correctly for the viewport
- Has a `<meta name="viewport">` tag with `width` or `initial-scale`
- Contains some content when JavaScript is not available
- Provides a valid `apple-touch-icon`

Additional items to manually check (3) — These checks are required by the baseline PWA Checklist but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

PWA Lighthouse

You can find the code of static and dynamic caches here.

Which one should I use static or dynamic?

It's up to your needs and your projects. I prefer to use the dynamic cache when I don't have API calls. When I have API calls, I prefer to choose by my self which resources to store with the static cache.

You can read more about **PWA** on google developers here.