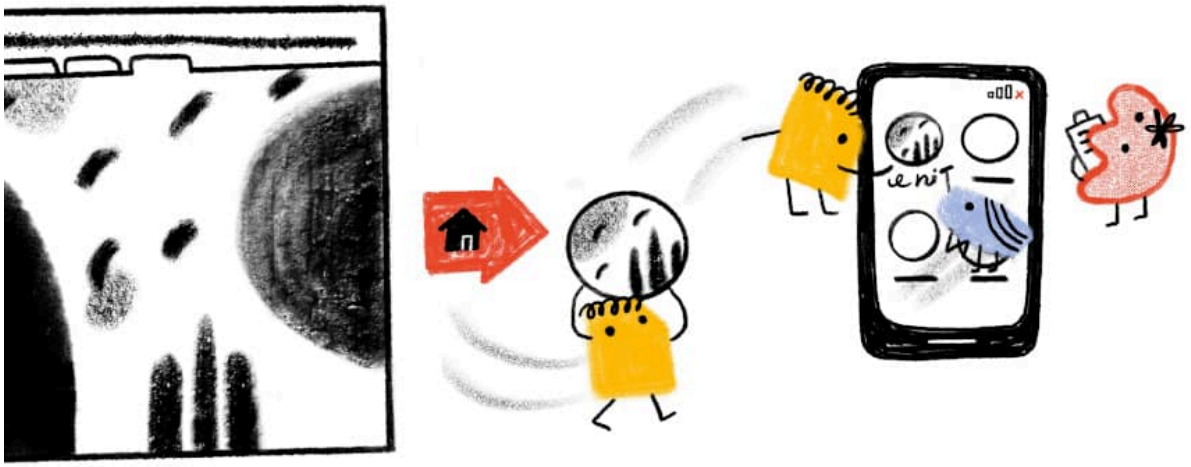
PWA 2025



Content team

*Hello content team! This is your personal doc to collaborate on and plan the contents of your chapter. Click **Request edit access** above to get started.*

Please add your name and email address below so we can @tag each other in the comments. You can also subscribe to all comments by opening the comment history, clicking the notification bell, and selecting All.

***Authors:** …*
***Reviewers:** …*
***Analysts:** …*

The objective of your chapter is to write a data-driven answer to this big question:

"What is the state of PWA in 2025?"

Learn more about the chapter lifecycle and refer to your chapter's tracking issue on GitHub for more info. Thank you all for your contributions! *Official call*

Outline

The purpose of this section is to define the scope of the chapter by creating an ordered list of all of the topics to be explored. You can think of this outline as the chapter's table of contents. This list will become your narrative, so consider how the content should be sequenced and how much additional depth is needed for major topics. You may choose to start with last year's outline and add or remove content as needed. Every chapter must have an introduction and conclusion, but everything in between is up to you.

Every chapter must also be data-driven, so for each topic in the outline below, clearly enumerate which metrics you'll need to substantiate your narrative. Work with your analysts to clarify what data is needed and how the results should be formatted. For example, if you're measuring the usage of a particular HTTP header value, you can measure it as the percentage of pages having that header, as the percentage of headers having that value, as a distribution of values, what the largest value is, etc. Clarify those expectations upfront so that the analysts know how to write the corresponding queries and whether the metrics are even feasible in the dataset.

First meeting to outline the chapter contents by June 1
Custom metrics completed by July 1
HTTP Archive crawl by July 1
Querying all metrics and saving the results by September 1
First draft of chapter by October 1
Reviewing & Editing of chapter by October 20
Publication of chapter (Markdown & PR) by November 15

PWA/Web Apps Chapter Outline

HTTP Almanac 2025

Introduction

The year 2015 was the first time we read about Progressive Web Applications. Nine attributes – responsive, connectivity independent, app-like-interactions, fresh, safe, discoverable, re-engageable, installable and linkable – were what defined the cutting edge of what could be achieved with web technologies back then. The PWA concept was coined 10 years ago, and it is with great pride that we look at the state of this set of technologies, one decade after its ideation.

The concept of a PWA has evolved a lot in these 10 years, and different browsers support it in different variations and with different names. An idea that started as a way of enabling access to a web application via “A2HS” (Add to home screen) on mobile browsers is now present on multiple platforms that allow to integrate web content directly into the underlying platform. Let’s dive into what the last couple of years have brought to PWAs.

Changes to PWA/Web Apps

The last couple of years have seen new features coming to web apps that enable more customization, more control of the application’s behaviour and better performance.

Another important change for web apps is the requirement that they must have for the browser to prompt in Chromium for an installation of the app. Whilst in the past an app was required to have a manifest file and a service worker, there is no need to have a service worker anymore. On the other hand, Safari allows any web page to be installed. Some browsers create a default offline experience that shields users from lack of connectivity, but bear in mind that service workers are still required for the best offline UX. For an up-to-date summary of web app installability, see this MDN article.

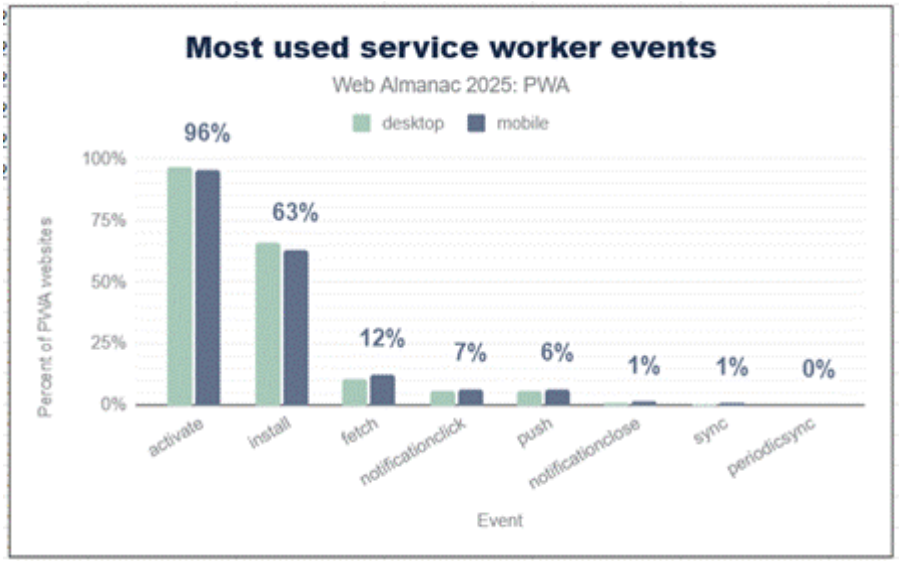
With this summary, we are now ready to jump into the data and understand the state of PWAs right now. We will finalise by comparing this year's data with the data from 3 years ago, which was the last time there was a chapter on PWAs, when possible.

Service Worker

Service Workers (SW) remain essential to allow advanced capabilities like background sync, offline support and push notifications to web apps. This year, the data suggest around one fifth of web properties using Service Workers. These are the most used capabilities of Service Worker:

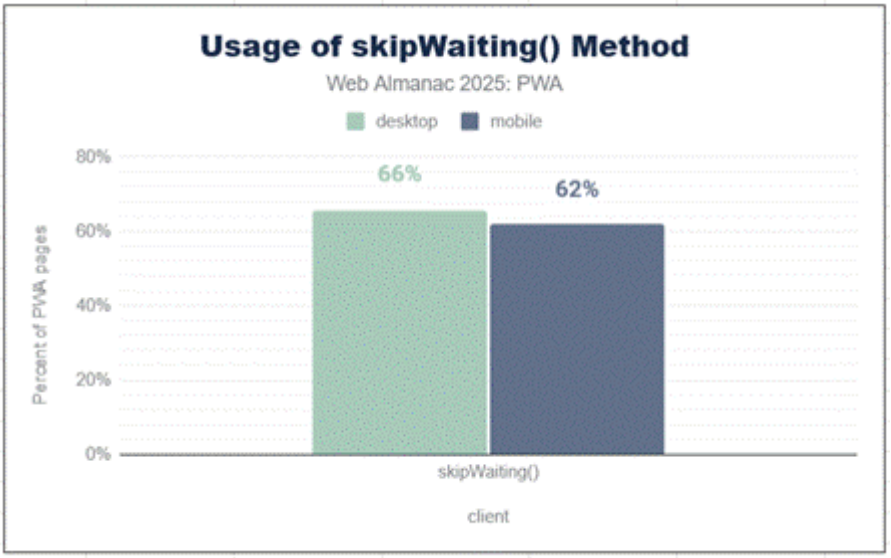
Service Worker Events

The most used event for these Service Workers is the activation with almost every SW using it, with around 96% of PWAs using it. The `install` event takes second place with around 64% usage. This might suggest that applications are caching resources to speed up their loading times and performing SW management. Usage of other advanced events, like `fetch`, `notificationclick` and `push` falls considerably.



Service Worker Methods

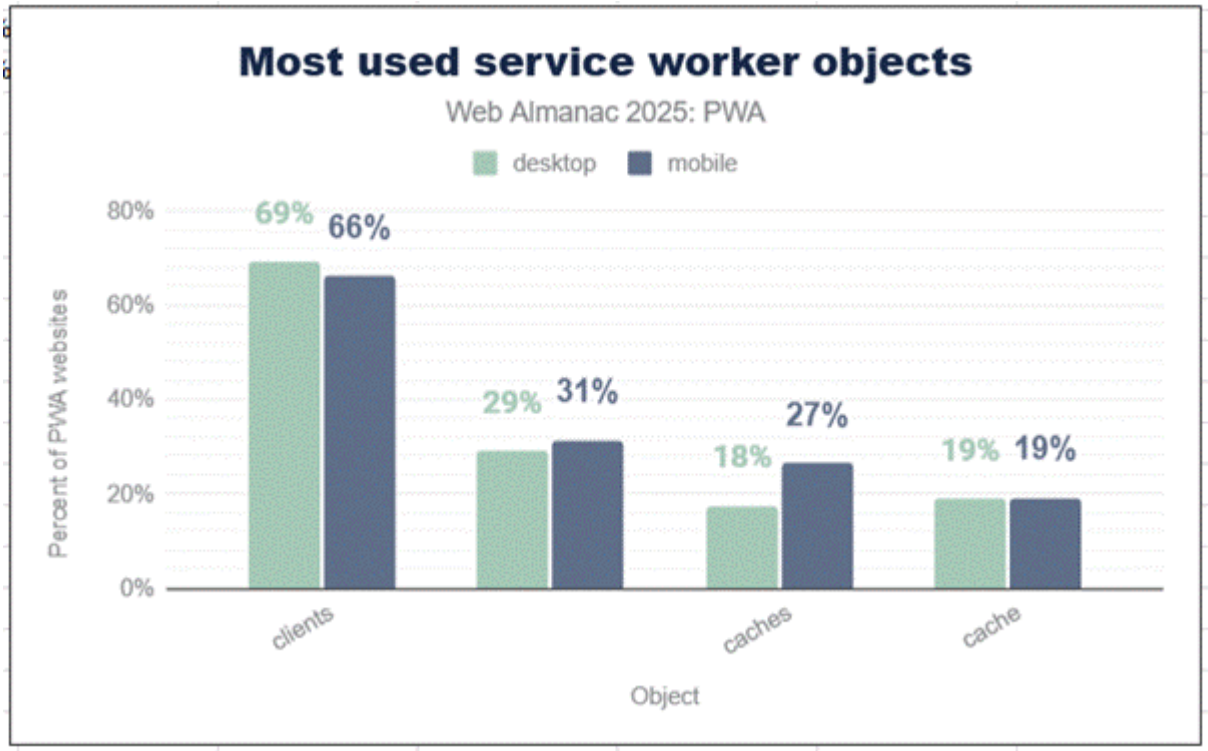
Looking at the most used Service Worker methods, `skipWaiting()` has a notable use, with 66% usage on desktop and 62% on mobile. The browser will activate the Service Worker and replace the old one immediately. This can help users get the latest version of the application right away.



Service Worker objects

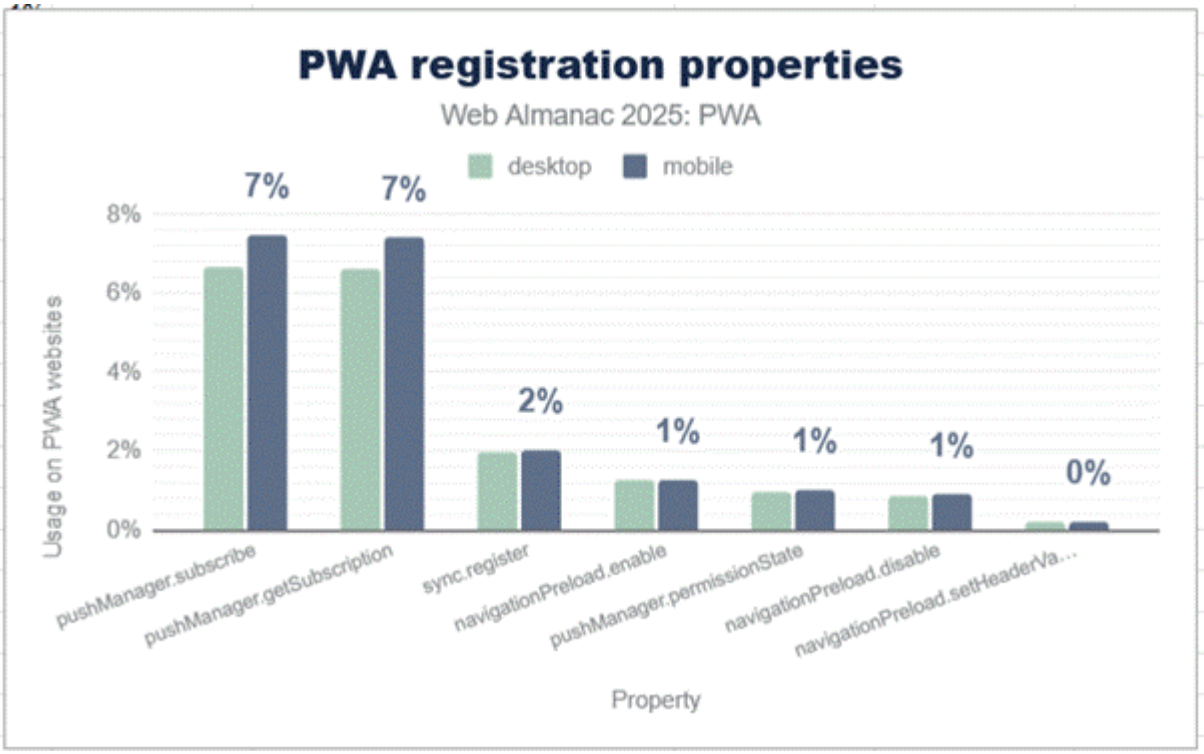
The most used SW objects are `clients`, `caches` and `cache`. For clients, this can be expected as it is the way to tell the Service Worker to take control of all open pages by calling `clients.claim`. Regarding `caches`, management methods appear on top, also not surprising considering that these are the methods that developers would use to make sure their assets are up to date to achieve that speedy page load.

As hinted before, the main methods from these correspond to `claim`, `open/delete/keys/match`, and `add`.



Registration Properties

Diving deeper into service worker functionality, the data sheds some light into advanced capabilities that PWAs are using. For all PWAs using Service Workers across desktop and mobile, ~7% are registering for pushManager, 2% register for sync and 2% register for navigationPreload.



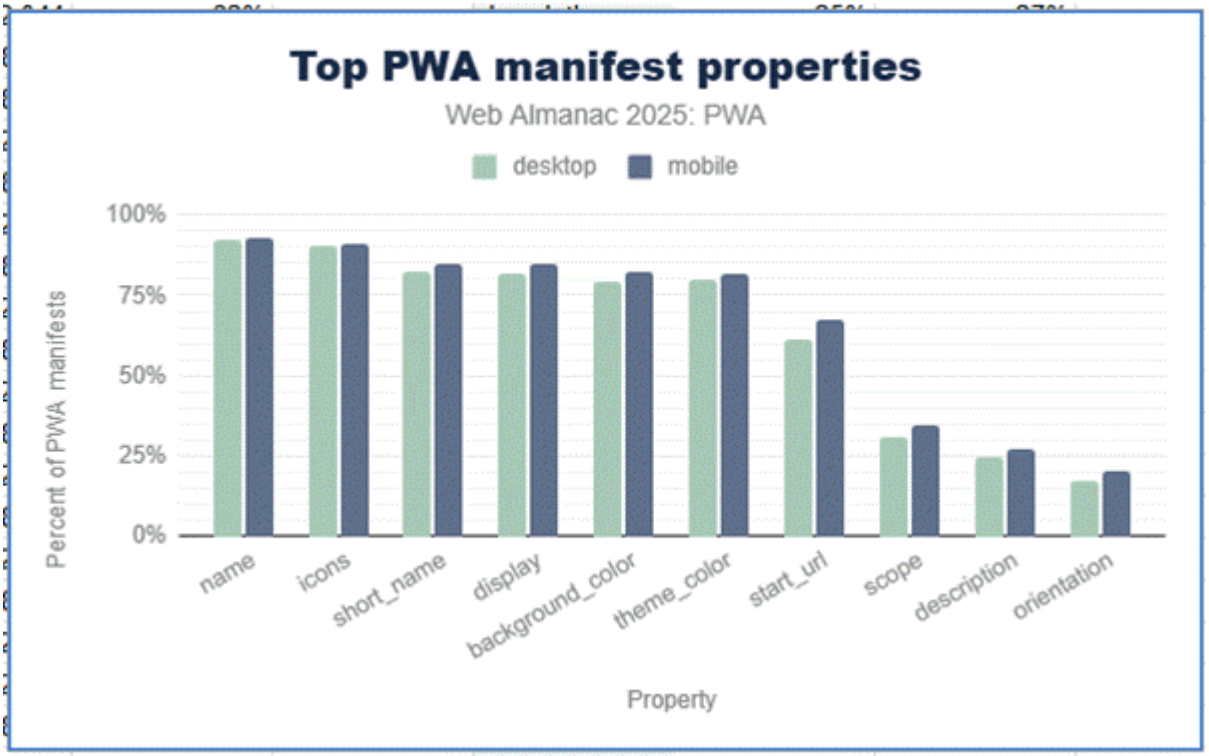
Web App Manifest

The web application manifest is now, more than ever, the most important part of a web app. It defines a look and feel, enables advanced capabilities that are gated behind an installation and is becoming an integral part that identifies a web application as a whole. But in order to be effective, the manifest file needs to be well formed. For the current year, 94.4% of desktop sites and 95% of mobile sites are parseable. There is no change from the last data set, and same as last time around, the fact that the manifest file was able to be parsed does not imply completeness or minimum availability of features. Many values in the manifest, as important as they may seem, have reasonable fallbacks in place.

From those parseable manifests, we will now look at individual present fields. This can give us an understanding of how developers are using the manifest file and if there have been changes since 2022.

Manifest properties

Straight up, these are the most used PWA manifest properties: name, icons, short_name and display and background_color. The top 4 most used properties are the same ones from 2022, with subtle notable changes regarding their order.

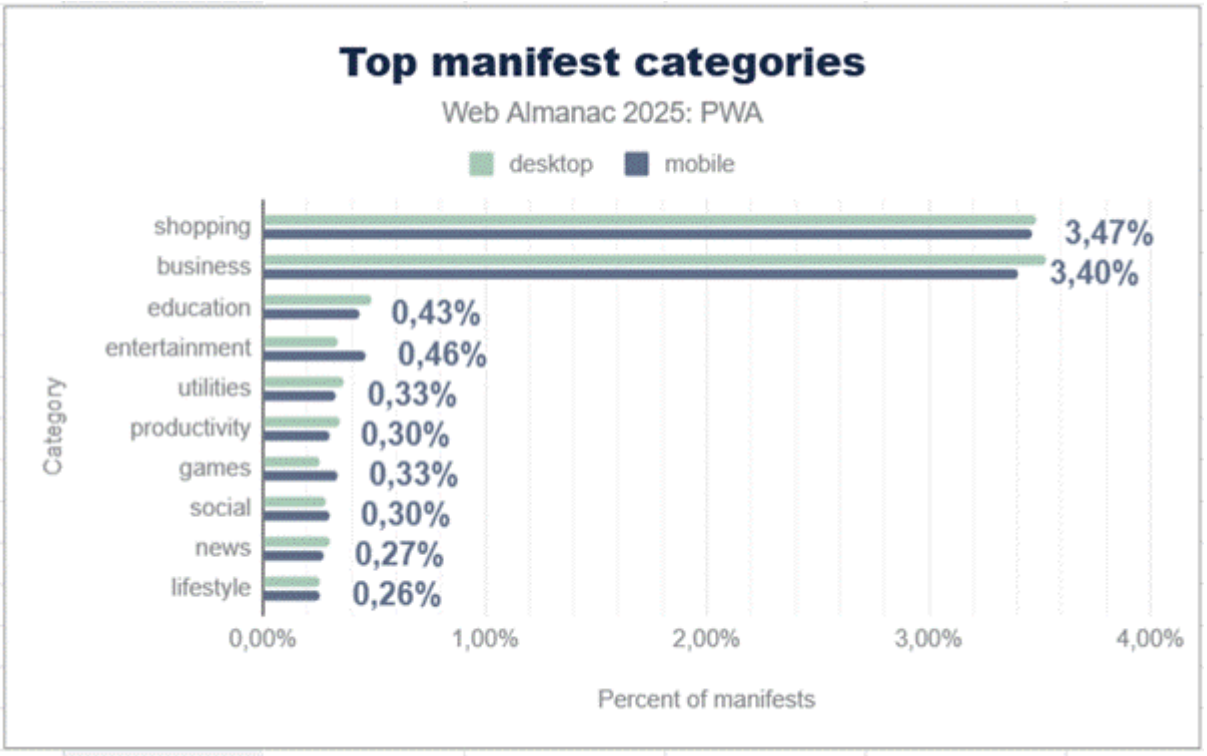


Let's examine how individual members rate in the totality of manifest files scanned by the almanac. Unless noted otherwise, values are very similar so I will refer to both mobile and desktop sites.

Manifest field	**Percentage of manifests that have it**
`description`	24%
`file_handlers`	0.1%

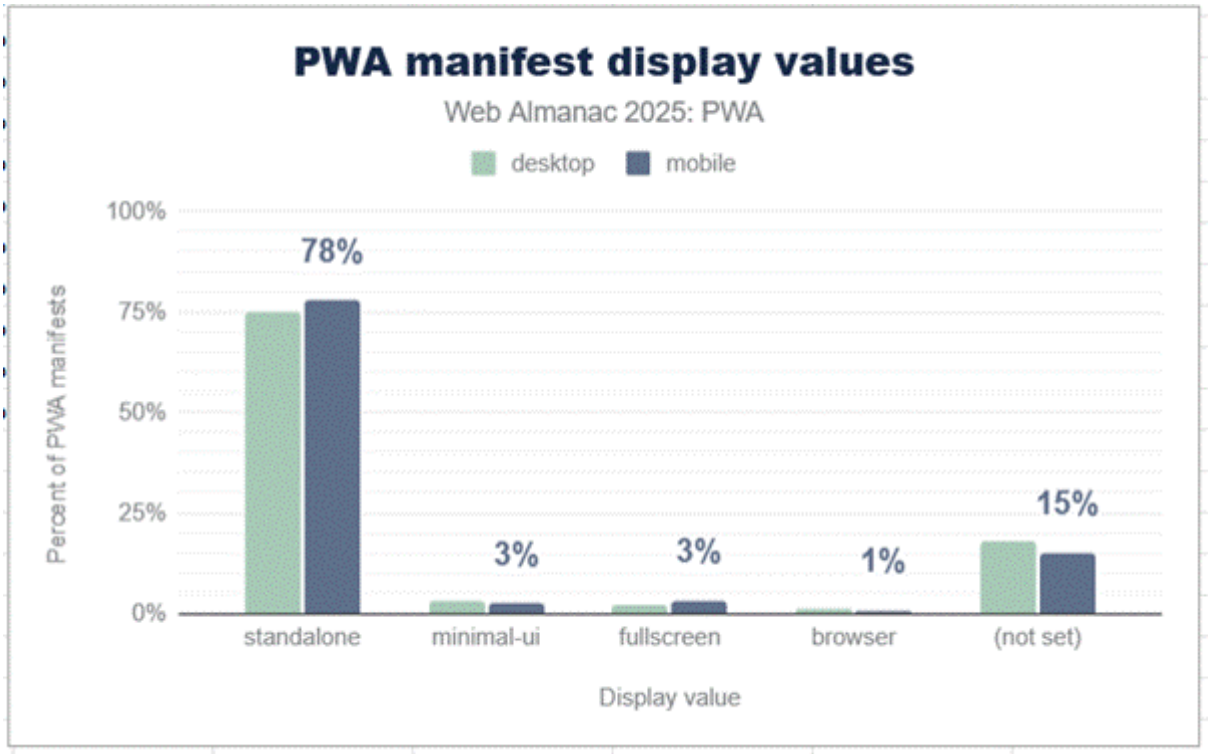
`iarc_rating_id`	0.1%
`lang`	14%
`screenshots`	2.7%
`share_target`	1%
`shortcuts`	7%
window-controls-overlay	0.046%
`note-taking`	0.07%
`protocol_handlers`	0.21%
`prefer_related_applications`	4%

For the manifests that specify the `categories` member, the top categories are:



Manifest `display` values

The display member is used to specify the preferred display mode for the web app. Different browsers have different interpretations of these values, but overall it hints to the UA how much of the browser UX to show/hide.



Most web apps (78%) opt for a `standalone` value for the display member. Standalone is common because it makes the app look more native-like.

Manifest `icons` sizes values

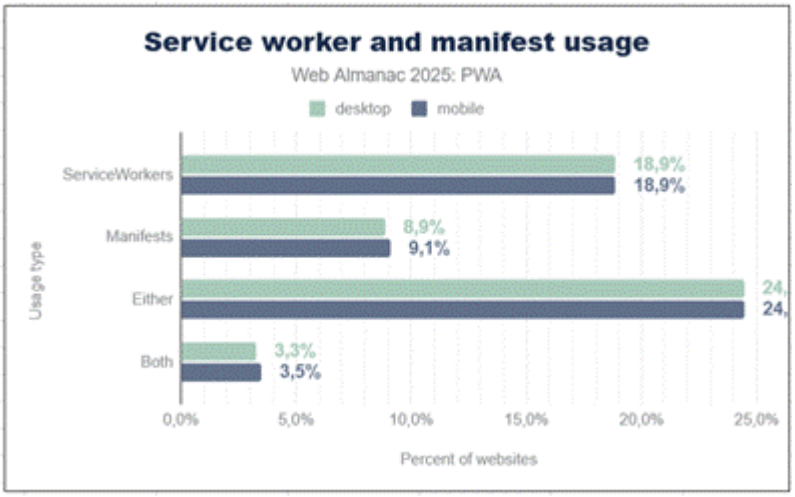
Top sizes include 192x192 and 512x512.

Manifest `orientation` values

Unsurprisingly, around 79.5% of PWAs do not set orientation. Responsive design and modern development make defining the app's orientation less necessary, but portrait takes second place with around 11.9%.

Service Worker and Manifest Usage

We've seen the latest data on what the most used Service Worker and Manifest features are. In 2025, we can see that roughly one fifth of sites use Service Workers, and roughly one tenth use Manifests.



Overall, there are considerable changes to the data from the 2022 HTTP Almanac. Service Worker usage has taken a huge leap from around 1.7% to 18.9%. That is around a 10 times increase in adoption. Manifest usage is slightly higher but remains at a very similar spot as it was 3 years ago.

PWAs and FUGU APIs

These are the top 10 used advanced capabilities in PWAs for 2025.

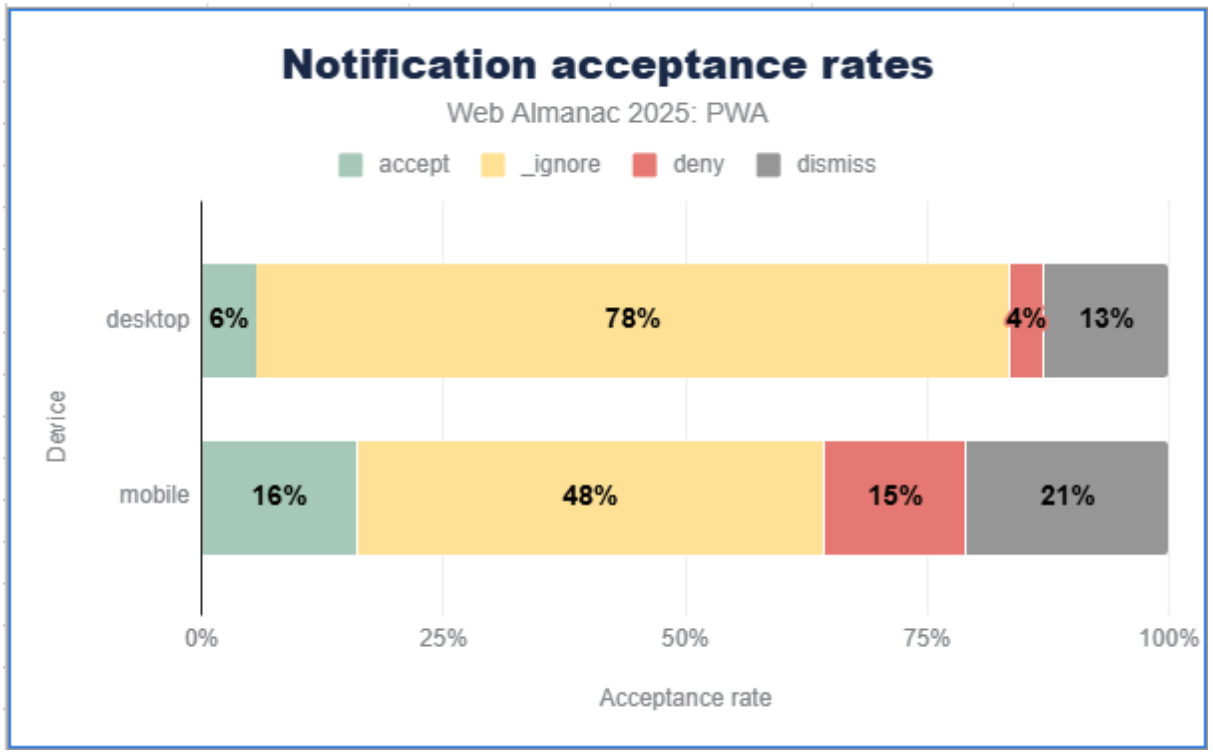
Capability	**% Mobile PWAs**	**% Desktop PWAs**
Compression Streams	18.6	21.1
Async Clipboard	17.9	19.2
Device Memory	10.3	10.3
Cache Storage	9	3
Web Share	9	8

Media Session	6.8	7.8
Media Capabilities	6.4	7.4
Add to Home Screen	6.8	7.3
Service Worker	3.7	3.3
Push	1.7	1.6

There is a complete separate chapter dedicated to capabilities to dive deeper in the adoption that these sort of APIs have had in 2025.

Notifications and PWAs

Many applications will try to get the user to accept notifications. This is a powerful way to allow the user to re-engage with the application. This is a controversial capability as there is considerable bad UX and dark patterns to try to get users to accept them. The data shows that in both desktop and mobile, the most common action a user takes is to ignore these requests.



Conclusion

It has been a decade since PWAs. The landscape has changed a bit as the technology and capabilities reach maturity; we now have all major browsers (Edge, Chrome, Firefox and Safari) supporting to a degree web apps. Web apps continue to show growth and almost one fourth of web sites crawled have either a Service Worker or a Manifest file.

Advanced capabilities continue to be used sparingly, likely due to slow implementor support, but the oldest capabilities like Web Share are starting to show up

As we close the 2025 Web Almanac PWA chapter, there's ongoing work and agreement towards looking at a way to democratise web app distribution by baking installation

capabilities directly into the platform. We hope in the next 10 years the PWA technologies will follow the same fate as responsive design, where they have been commoditized and web apps that have good UX for application lifecycle management will be the norm, in conjunction with newer capabilities that redefine what a web app can do. Here’s to the next decade for web apps!