Engineer

Progressive web app

a website that can be installed and run like an app

What it is

A progressive web app, also written Progressive Web App and usually shortened to PWA, is a website built so that a browser can install it and run it like an app: from its own icon, in its own window, and often with part of it still working when the network is gone. It stays an ordinary website the whole time. Nothing comes from an app store, and the pages are the same HTML, CSS and JavaScript a browser would load in a tab.

No standard defines the term. What the standards define are the two parts it is built from. The first is the web app manifest, a JSON file described by the W3C's Web Application Manifest, which gives the app a name, icons, a start URL and a display mode. That specification sets a low bar for who qualifies: "Any website is an installable web application." A browser can offer a way to install one, and the installed app then opens with the manifest's settings applied. In the `standalone` display mode it is meant to resemble a native application, which can mean a window of its own and an icon in the application launcher, and the browser leaves out its standard interface, such as the URL bar.

The second part is the service worker, from the W3C's Service Workers specification. It is a script the browser runs apart from the page, which sits between the page and the network, sees the requests the page makes and can answer them from caches the script controls. That is what lets a web app keep serving its pages while offline. Those caches belong to the script: the specification keeps them completely separate from the browser's ordinary HTTP cache, and they change only when the script changes them.

The Web Application Manifest never uses the word "progressive", and the Service Workers specification uses it once, in a note about how Chromium recognises a PWA. The Next.js guide to PWAs recommends progressive enhancement, so that

the app still works well in a browser that does not support some of its PWA features. Every part is optional in that sense. A browser that ignores the manifest, or has no service worker support, still shows the website.

A progressive web app is not a separate kind of website either. A single-page application, a static site and a server-rendered one can each add a manifest and a service worker. The term describes what a browser can do with a site after it has loaded, not how its HTML is produced.

Use the concept in a decision

Put “What is a progressive web app?” to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The Next.js guide makes the case in three points. Updates go out at once, without waiting for app store approval. One codebase serves every platform. And the app gets features people expect from native apps, such as installing to the home screen and push notifications. All three follow from the app still being a website: a new version can be published without a separate app-store release; when each device adopts it depends on caching and service-worker updates.

Offline is the reason the service worker exists. The specification’s own motivation section says that web applications have traditionally assumed the network is reachable, and that other application platforms consequently cope better with interrupted connections. Service workers give web applications a way to handle that interruption too. With one in place, a train in a tunnel or a ship out of range of the coast no longer has to end in the browser’s error page.

The two halves are independent. The Next.js guide notes that install prompts do not need offline support, and the manifest specification has removed its own `serviceworker` member. An app can be installable with no service worker, and a service worker can make an ordinary site work offline with no manifest at all.

What both halves share is HTTPS. Next.js lists a site served over HTTPS as one of two things an app needs before it can be installed on a mobile home screen, the other being a valid manifest. The Service Workers specification says service workers must run in secure contexts, which in practice means HTTPS, with localhost allowed for development.

What it looks like in practice

Three small files make a working PWA, all written for this article. The manifest names the app, picks the display mode and lists its icons:

```
{
  "name": "Field Notes",
  "short_name": "Notes",
  "start_url": "/",
  "display": "standalone",
  "icons": [
    { "src": "/icon-192.png", "sizes": "192x192", "type": "image/png" },
    { "src": "/icon-512.png", "sizes": "512x512", "type": "image/png" }
  ]
}
```

Each page links the manifest and registers the service worker, but only in a browser that supports it:

```
<link rel="manifest" href="/manifest.webmanifest" />
<script>
  if ("serviceWorker" in navigator) {
    navigator.serviceWorker.register("/sw.js");
  }
</script>
```

And the service worker keeps a small set of files for offline use:

```
// sw.js
const CACHE = "shell-v2";
const SHELL = ["/offline.html", "/app.css", "/app.js"];

async function precache() {
  const cache = await caches.open(CACHE);
  await cache.addAll(SHELL);
```

```
}

async function dropOldCaches() {
  const names = await caches.keys();
  const old = names.filter((name) => name !== CACHE);
  await Promise.all(old.map((name) => caches.delete(name)));
}

async function respond(request) {
  if (request.mode === "navigate") {
    try {
      return await fetch(request);
    } catch {
      return caches.match("/offline.html");
    }
  }
  const hit = await caches.match(request);
  return hit ?? fetch(request);
}

self.addEventListener("install", (event) => {
  event.waitUntil(precache());
});

self.addEventListener("activate", (event) => {
  event.waitUntil(dropOldCaches());
});

self.addEventListener("fetch", (event) => {
  if (event.request.method !== "GET") return;
  event.respondWith(respond(event.request));
});
```

The manifest uses `.webmanifest`, the file extension registered for it, and is served with the media type `application/manifest+json`. The service worker sits at the root because its scope defaults to the directory the script is in, so a worker at `/js/sw.js` could only control pages under `/js/` unless the server sent a `Service-Worker-Allowed` header. When it installs, it opens a cache named `shell-v2` and stores the offline page, the stylesheet and the script. `addAll()` rejects if any of those files comes back with an error status, and because `waitUntil()` was handed that promise, the whole installation fails with it instead of leaving a half-filled cache behind. On activation it deletes every cache with another name. Page navigations go to the network first and fall back to the cached offline page when the network does not answer. Everything else is served from the cache when it is there and fetched when it is not. Requests other than `GET` are left alone.

The company describes a full build of this on a real platform. The portfolio entry explains why it was needed: many of the platform's users are seafarers and field staff on phones, often offline or on a weak connection, so the app had to install like a native one and keep working when the signal drops. It covers both halves. The app installs to the home screen with icons in several sizes, a splash screen and theme colours. The service worker is set up through the next-pwa plugin, and Workbox runtime caching serves fonts, images, audio, video and CDN assets cache-first, since none of them change from one request to the next, alongside HTTP cache-control headers. Because service workers need HTTPS, the install and offline behaviour were tested over HTTPS on real devices. The outcome it reports is an app people install and go on using offline, with repeat visits loading from the cache.

The install half also looks different from the reader's side. The site's page on installing it as an app walks through where each browser keeps the command: the Share menu and Add to Home Screen in Safari on an iPhone, a small icon on the right of the address bar in Chrome, Edge, Brave, Opera and Vivaldi on a computer, and Add to Dock in Safari on macOS Sonoma or later. It also states that iOS never suggests installing on its own. Firefox provides separate web-app windows on Windows from version 143, or version 150 for Microsoft Store installations, but not on macOS or Linux. The page is also clear about what installing this site gives: an icon and a window, but no offline copy, because offline support needs a service worker.

Where it goes wrong

Users keep the old version. According to the Service Workers specification, cache objects are not updated unless the author explicitly asks for it, and they do not expire unless the author deletes them. A cache-first worker therefore serves yesterday's stylesheet until its script says otherwise. Put a version in the cache name, change it whenever the cached files change, and delete the old caches on activation, as the example does. Changing the script is what starts an update, because the browser compares the newly fetched `sw.js` byte for byte with the installed one. Even then the new worker waits until no tab or window is using the old one, unless it calls `skipWaiting()`, so a user who keeps a tab open runs the previous version for as long as that tab stays open. The Next.js guide serves `sw.js` with `Cache-Control: no-cache, no-store, must-revalidate`, so that users always get the latest version of the worker itself.

It works on the laptop and not on the test phone. `navigator.serviceWorker` exists only in secure contexts. On a page served over plain HTTP the usual feature check is quietly false, the worker is never registered, and nothing on the page says so. The specification lets browsers allow localhost, 127.0.0.0/8 and ::1/128 for development, which is why a worker that runs on a developer's own machine can be missing when a phone opens the same app over HTTP from that machine's address on the local network. Next.js offers `next dev --experimental-https` for local testing over HTTPS. Test install and offline behaviour on real devices, over HTTPS, before calling either one done.

A fetch listener that does nothing. The Service Workers specification records that some sites register a `fetch` listener with an empty body only so that Chromium will recognise them as a progressive web application. The specification now detects listeners like `() => {}`, and it encourages browsers to warn that such listeners are unnecessary and may hurt performance. Write a handler that answers requests, or leave the listener out.

Counting on an install prompt. Whether to offer installation, and how, is up to the browser. Next.js says modern browsers show an install prompt automatically once a site has a valid manifest and HTTPS, and advises against building a custom install button on `beforeinstallprompt`, because that event does not work across browsers and platforms, Safari on iOS included. The manifest specification has since removed the event altogether. The display mode and start URL are advisory as well: a browser that does not support `standalone` falls back to `minimal-ui` and then to `browser`, an ordinary tab. Tell users how to install, as the iOS instructions in the Next.js guide do, and design for the app being opened in a tab.

Further reading

- Web Application Manifest — the manifest format, its members, display modes, and what installing and uninstalling mean.
- Service Workers — the worker's life cycle, scope, update rules, the secure-context requirement and the Cache API.
- Web Application Manifest: Installable web applications — how a browser installs a web application, launches it and lets the user remove it again.
- Service Workers: Understanding cache lifetimes — why cache objects never update or expire unless the script tells them to.
- How to build a Progressive Web Application (PWA) with Next.js — a manifest file, push notifications, install instructions for iOS, HTTPS in development and security headers for the service worker.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a progressive web app?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/progressive-web-app/