# Assets and data


- Platform-specific apps vs. PWA
- Caches and storage
  - The need for service workers
- Offline-ready
  - Frequently used cache approaches
  - Updating assets
- Size and Lifespan
- Resources


Developing an application typically involves several assets and resources, from the logic and code (compiled or not) to the user interface elements such as screen designs, images, logos, and fonts.

A Progressive Web App is a website, so all its assets are the same as on the web:

- HTML for content and initial page rendering.
- JavaScript for logic, including the ability to run WebAssembly modules (compiled code) and render 2D and 3D hardware-optimized graphics.
- CSS for layout, styling, and animations.
- Images in web formats, such as PNG, JPEG, WebP, and SVG.
- Videos in web formats, such as MPEG-4 and WebM.
- Web fonts.
- Data in JSON or other formats.

By default, websites download assets over the network, starting with HTML and continuing to the rest of the resources.

Managing those resources to load fast and be available offline has been a challenge for the web. Nowadays, PWAs use capabilities previously associated only with platform-specific apps.

## Platform-specific apps vs. PWA

When you install a platform-specific app you are typically downloading a package that includes all the app's assets: code, images, fonts, videos, and so on. Therefore, all these resources are

available, from your device storage, even when the app is offline.

On the other hand, a classic website needs a network connection to download the assets when required. If you are offline, you will see an error from the browser as there are no assets client-side.

The PWA approach enhances the traditional web experience by making some or all the assets available client-side as with platform-specific apps. Therefore, when you open a PWA, the initial rendering can be as instantaneous as a platform app because the assets are available without going to the network.

{% Aside %} An app that is available offline will always render a basic user interface instead of a technical error from the operating system or browser. Some apps still offer all their functionality, some a limited experience, and others a simple message with the app's branding. This applies to both platform-specific apps and PWAs. {% endAside %}

# Caches and storage

Since the beginning of the web, developers have not had complete control of how a resource is cached. The browser is in charge of the HTTP cache and it may or may not cache and serve resources based on different policies. Other storage options like Web Storage and IndexedDB were meant to save simple data and objects.

PWAs don't need to rely on those policies alone for their content. Instead, we have solutions today to gain better control over when and how to cache resources and when and how to deliver them locally: the Cache Storage API. The web has a few of client-side storage solutions available:

- **Web storage**: Includes `localStorage` and `sessionStorage`. These APIs store simple key/value string pairs. Web storage is limited, and has a synchronous API, so it should be avoided whenever possible.
- **IndexedDB**: An object-based database (No-SQL) with an asynchronous API that is good for web performance. IndexedDB can store structured and binary data client-side. It is typically what you will use to store data, like what you'd get from or send as an API request. It's also useful to save data locally immediately and later on sync it to the server.

  The IndexedDB API is used to interact with the database.
- **Cache storage**: A collection of HTTP request and response pairs you can use to store and retrieve resources — with their HTTP headers — exactly as they are delivered from the server. The Cache Storage API allows you to store, retrieve, update and delete network requests, for example for your assets, even when offline.

{% Aside 'warning' %} You may have heard about WebSQL and ApplicationCache. These two solutions are deprecated, and you shouldn't use them in your PWA. {% endAside %}

# The need for service workers

A PWA can store its assets in Cache Storage and IndexedDB, but how can we use those assets to deliver a fast and offline experience to your users. The answer? Service workers.

With a service worker, you can serve assets without going to the network, send notifications to a user, add a badge to your PWA icon, update content in the background, and even make your whole PWA work offline. Learn more about service workers in the next chapter.

# Offline-ready

Users expect your application to offer a fast and always-ready experience. That means your app should work offline.

Being offline-ready doesn't mean that all your content or services should be available without a network connection. However having at least a basic experience when the user is offline, like a page that asks you to connect to the Internet to continue, will provide for a better user experience, keeping the user within the experience of your app instead of a generic browser error. In some browsers this is a must-have feature to pass the PWA installation criteria. Displaying your PWA's user interface, along with cached content, is better. Letting users continue using your whole PWA and syncing server changes when they're back online is the gold standard for working offline.

To make your app available offline, you will need to cache the assets necessary for your offline experience and make your service worker serve them later. Make sure to add the offline assets to your cache before you need them. This is a particular case where you cannot cache them when requested.

{% Aside %} Asset caching happens independently from PWA installation, and will work even from a browser tab, allowing for faster asset delivery and an offline experience regardless of how a user chooses to use your PWA. {% endAside %}

# Frequently used cache approaches

In your PWA, you are in charge of all the decisions. You can choose the best approach to cache or install assets based on your needs. Some decisions to make are:

- Precaching: pick the assets you'd like to install on the first load; those should include the HTML and the minimal assets to render the app. When using precache keep in mind you are using the device's space and network. Be conscious and responsible when downloading assets and caching them.

- Cache as needed: used to add assets to the cache when they are requested. Typically these are assets that can change independently of your app version, like images or content. See the caching section for different strategies on how to cache as needed.
- APIs and web services: API calls can be cached; or, instead of caching the API responses, you can store their data in IndexedDB.

{% Aside 'gotchas' %} With platform-specific apps, the most common pattern is to include all the app's assets in its package; that's why these apps are commonly larger than 500 MB. On the other hand, A PWA can initially install only the minimum set of resources to offer an offline and fast experience and download the rest later, with PWAs using considerably less storage on installation. {% endAside %}

# Updating assets

In the standard app model for app catalog installed apps, when developers need to update their app, they publish a new package. Users need to download the whole package again, even if most of the assets haven't changed. With PWAs, using the approaches on the section above, you decide how and when to update assets. Here are different options for how to update assets:

- Full Update: in this case, every time you make a change in the app, even a minor one, your code will download all the assets again in the cache.
- Partial Update: in this approach, you create a method to define which assets have been updated, and you update only those, with or without the user's intervention.
- Continuous Update: using this technique, your assets will be checked and updated automatically on every PWA usage individually

{% Aside %} When caching assets, you need to decide on a cache swap or clean up policy that defines when the user will see an updated asset; taking into account storage usage and the balance between a fast experience and displaying a slightly stale asset. {% endAside %}

# Size and Lifespan

Usually platform-specific apps don't deal with size limits; at installation, apps can be gigabytes or more in size. As long as the device has the capacity the installation will be allowed. Also, while the app is installed, it will use that total amount of storage no matter if you use the app or not. Storage is handled differently for PWAs. The browser will store your assets based on policies that you define in your PWA's logic.

{% Aside %} The users are always in control, and they can delete the assets at any time. For PWAs, manual eviction happens via the browser's settings. {% endAside %}

Size limits depend on the browser. It's not as flexible as for platform-specific apps, but it's typically good enough for most web apps. You can find the specific limitations by browser in this article.

Browsers can evict assets based on storage pressure, or after some weeks of inactivity, if the user is using your PWA in the browser. On some platforms, if the user installs your PWA, eviction won't happen. Where available, code can request persistent storage through an API to avoid that eviction.

{% Aside %} If storage eviction happens for any reason and the user installed your PWA, the PWA will still work if you have a network connection. When you open that app again, the browser will start the service worker lifecycle again, loading your app from the network. {% endAside %}

# Resources

- Storage on the Web
- MDN: Web Storage
- MDN: IndexedDB

---

□□□ #2
□ □□□ □□□ 2026-10-31 22:54:58 UTC
□ □□□ □□□ 2026-10-31 22:54:59 UTC