Building multiple Progressive Web Apps on the same domain

- Technical terms
- The case for multiple, related PWAs
- Using separate origins
- Using the same origin
  - Non-overlapping paths
  - Overlapping/nested paths
- Challenges for multiple, same-origin PWAs
- Additional challenges for overlapping/nested paths
- Conclusion
- Additional resources

In the Progressive Web Apps in multi-origin sites blog post, Demian discussed the challenges that sites built on multiple origins face when trying to build a single Progressive Web App that encompasses all of them.

An example of this type of site architecture is a ecommerce site where:

- The home page is at `https://www.example.com`.
- The category pages are hosted at `https://category.example.com`.
- The product detail pages at `https://product.example.com`.

As discussed in the article, the same-origin policy imposes several restrictions, preventing the sharing of service workers, caches, and permissions across origins. For that reason, we strongly recommend avoiding this type of configuration and for those that already have sites built in this way, to consider migrating to a single origin site architecture whenever possible.

Diagram showing a site divded into multiple origins and showing that technique is discouraged when building PWAs.

Avoid using different origins for site sections of the same site when trying to build a unified Progresive Web App.

In this post, we take a look at the opposite case: instead of a single PWA across different origins we'll analyze the case of companies that want to provide **multiple PWAs**, taking advantage of the same **domain name**, and make the user aware that those PWAs belong to the same organization or service.

As you might have noticed, we are using different, but interrelated terms, like domains and origins. Before moving forward, let's review these concepts.

Technical terms

- **Domain:** Any sequence of labels as defined in the Domain Name System (DNS). For example: `com` and `example.com` are domains.
- **Hostname:** A DNS entry that resolves to at least one IP address. For example: `www.example.com` would be a hostname, `example.com` could be a hostname if it had an IP address, and `com` would never resolve to an IP address and so it could never be a hostname.
- **Origin:** A combination of a scheme, hostname and (optionally) port. For example, `https://www.example.com:443` is an origin.

As its name implies, same-origin policy imposes restrictions on origins, so we'll mostly refer to the term throughout the article. Nevertheless, we'll use "domains" or "subdomains" from time to time, to describe the technique being used, in order to create the different "origins".

The case for multiple, related PWAs

In some cases, you might want to build independent apps, but still identify them as belonging to the same organization or "brand". Reusing the same **domain name** is a good way of establishing that relationship. For example:

- An ecommerce site wants to create a standalone experience to let sellers manage their inventory, while making sure they understand that it belongs to the main website where the users buy products.

- A sports news site wants to build a specific app for a major sporting event, to let users receive stats about their favorite competitions via notifications, and install it as a Progressive Web App, while making sure that users recognize it as an app built by the news company.
- A company wants to build separate chat, mail, and calendar apps and wants them to work as individual apps, tied to the company's name.

> *Avoid using different origins for site sections of the same site when trying to build a unified Progresive Web App.*
>
> Company that owns example.com wants to provide three independent apps or PWAs, using the same domain name to establish the relationship between them.

Using separate origins

The recommended approach in cases like these is for each conceptually distinct app live on its own origin.

If you want to use the same domain name inside all of them, you can do that by using subdomains. For example, a company that provides multiple internet apps or services can host a mail app at `https://mail.example.com` and a calendar app at `https://calendar.example.com`, while offering the main service of their business at `https://www.example.com`. Another example is a sports site that wants to create an independent app completely dedicated to an important sports event, like a football championship at `https://footballcup.example.com`, that users can install and use independently of the main sport site, hosted at `https://www.example.com`. This approach might also be useful for platforms that let customers create independent apps of their own under the company's brand. For example, an app that lets merchants create their own PWAs at `https://merchant1.example.com`, `https://merchant2.example.com`, etc.

Using different origins ensures isolation between the apps, which means that each of them can manage different browser features independently, including:

- **Installability:** Each app has its own Manifest and provides its own installable experience.
- **Storage:** Each app has its own caches, local storage, and basically all forms of device-local storage, without sharing them with the others.
- **Service Workers:** Each app has its own service worker for the registered scopes.
- **Permissions:** Permissions are also scoped by origins. Thanks to that, users will know exactly which service they are giving permissions for, and features like notifications will be properly attributed to each app.

Creating such a degree of isolation is the most desirable in the use case of multiple, independent PWAs, so **we strongly recommend this approach**.

If apps on subdomains want to share local data with each other they will still be able to do it via cookies, or for more advanced scenarios they could consider synchronizing the storage through a server.

> *ALT_TEXT_HERE*
>
> Building different PWAs in distinct origins, by using subdomains is a good practice.

Using the same origin

The second approach is building the different PWAs on the same origin. This includes the following scenarios:

Non-overlapping paths

Multiple PWAs or conceptual "web apps", hosted on the same origin, with non-overlapping paths. For example:

- `https://example.com/app1/`
- `https://example.com/app2/`

Overlapping/nested paths

Multiple PWAs on the same origin, one of whose scope is nested inside the other:

- `https://example.com/` (the "outer app")
- `https://example.com/app/` (the "inner app")

The service worker API and manifest format allow you to do either of the above, using path-level scoping. However, in both cases, using the same origin presents many problems and limitations, the root of which stems from the fact that the browser won't fully consider these to be distinct "apps", therefore **this approach is discouraged**.

> *ALT_TEXT_HERE*
>
> Using paths (overlapping or not) to provide two independent PWAs (“app1”, “app2”) under the same origin is discouraged.

In the next section, we analyze these challenges in more detail, and what can be done, if using separate origins is not an option.

Challenges for multiple, same-origin PWAs

Here are some practical issues common to both same-origin approaches:

- **Storage:** Cookies, local storage, and all forms of device-local storage are shared between apps. For that reason, if the user decides to wipe local data for one app, it will wipe all the data from the origin; there's no way to do this for a single app. Note that Chrome and some other browsers will actively prompt users to wipe local data when uninstalling one of the apps, and this will affect data for the other apps on the origin as well. Another issue is that apps will also have to share their storage quota which means if either of them takes up too much space, the other will be negatively impacted.
- **Permissions:** Permissions are tied to the origin. That means if the user grants a permission to one app, it will apply to all apps on that origin simultaneously. That may sound like a good thing (not having to ask for a permission multiple times), but remember: if the user blocks permission to one app, it will prevent the others from requesting that permission or using that feature.
- **User settings:** Settings are also set per-origin. For example, if two apps have different font sizes, and the user wants to adjust the zoom in only one of them to compensate for it, they won't be able to do it without applying the setting to the other apps as well.

These challenges make it difficult to encourage this approach. Nevertheless, if you can't use a separate origin (e.g. a subdomain), as discussed in the Using separate origins section, from the two same-origin options we presented, using non-overlapping paths is strongly recommended, over overlapping/nested paths.

As mentioned, the challenges discussed in this section are common to both same-origin approaches. In the next section we'll go deeper into the details of why using overlapping/nested paths is the least recommended strategy.

Additional challenges for overlapping/nested paths

The additional issue with overlapping/nested paths approach (where `https://example.com/` is the outer app and `https://example.com/app/` is the inner app), is that all URLs in the inner app will actually be considered part of both the outer app and the inner app.

In practice this presents the following issues:

- **Installation Promotion:** If the user visits the inner app (for example, in a web browser), when the outer app is already installed in the user's device, the browser won't show the install promotional banners, and the BeforeInstallPrompt event won't be triggered. The reason is that the browser will check and see whether the current page belongs to an app that's already installed, and it will conclude that it is. The workaround for this is to install the inner app manually (via "Create Shortcut" browser menu option), or to install the inner app first, before the outer app.
- **Notification and the Badging API**: If the outer app is installed but the inner app is not, notifications and badges coming from the inner app will be erroneously attributed to the outer app (which is the nearest enclosing scope of an installed app). This feature works properly in the case where both apps are installed on the user's device.
- **Link Capturing**: The outer app may capture URLs that belong to the inner app. This is especially likely if the outer app is installed but the inner app isn't. Similarly, links within the outer app that link to the inner app will not link capture into the inner app, since they are considered to be within the outer app's scope. Additionally, on ChromeOS and Android, if these apps are added to the Play Store (as Trusted Web Activities), the outer app will capture all links. Even if the inner app is installed, the OS will still offer the user the choice of opening them in the outer app.

Conclusion

In this article we looked at different ways in which developers can build multiple Progressive Web Apps related to each other within the same **domain**.

In summary, we strongly recommend using a different origin (e.g. by using subdomains) to host independent PWAs. Hosting them in the same origin presents many challenges, mainly because the browser won't fully consider these to be distinct apps.

- Separate origins: **Recommended**
- Same origin, non-overlapping paths: **Not recommended**
- Same origin, overlapping/nested paths: **Strongly not recommended**

If it's not possible to use different origins, using non-overlapping paths (e.g. `https://example.com/app1/` and `https://example.com/app2/` it's strongly recommended over using overlapping or nested paths, like `https://example.com/` (for the outer app) and `https://example.com/app/` (for the inner app).

Additional resources

- Progressive Web Apps in multi-origin sites

With many thanks for their technical reviews and suggestions: *Joe Medley, Dominick Ng, Alan Cutter, Daniel Murphy, Penny McLachlan, Thomas Steiner and Darwin Huang*

Photo by Tim Mossholder on Unsplash

□□□ #3
□ □□□ □□□ 2026-10-31 21:42:00 UTC
□ cocobook-sync □□□ 2026-11-04 17:36:42 UTC