Web app manifest

- Adding a web app manifest to your PWA
  - Linking to your manifest
  - Debugging the manifest
  - For Chromium browsers
  - For Firefox
- Designing your PWA experience
  - Basic fields
    - Icons
      - Maskable icons
  - Recommended fields
    - Try it
    - Splash screens
  - Extended fields
  - Promotional fields
  - Capabilities Fields
- Resources

The web app manifest is a file you create that tells the browser how you want your web content to display as an app in the operating system. The manifest can include basic information such as the app's name, icon, and theme color; advanced preferences, such as desired orientation and app shortcuts; and catalog metadata, such as screenshots.

A web app manifest is a required installability criteria in every browser. Your PWA will not install without it.

Each PWA should include a single manifest per application, typically hosted in the root folder, and linked on all HTML pages your PWA can be installed from. Its official extension is `.webmanifest`, so you could name your manifest something like `app.webmanifest`.

While the recommended extension is `.webmanifest`, any filename will work as long as it's delivered with the `application/manifest+json` content type or another JSON-valid content type, such as

`text/json`. As such, many PWAs, particularly older ones, use `manifest.json` instead.

Adding a web app manifest to your PWA

To create a web app manifest, first make a text file with a JSON object that contains at least a `name` field with a string value: app.webmanifest:

```
{
   "name": "My First Application"
}
```

But creating the file is not enough, the browser needs to know it exists, too.

Linking to your manifest

To make the browser aware of your web app manifest, you need to link it to your PWA using a `<link>` HTML element and the `rel` attribute set to `manifest` on all of your PWA's HTML pages. This is similar to how you link a CSS stylesheet to a document.

index.html:

```
<html lang="en">
  <title>This is my first PWA</title>
  <link rel="manifest" href="/app.webmanifest">
```

If your PWA includes multiple HTML pages, make sure to link its manifest to all of them.

Debugging the manifest

To ensure the manifest is set up correctly, you can use Inspector in Firefox and DevTools in every Chromium-based browser.

If you see a message saying that the browser detected no manifest and you've included a `<link>` to it correctly, check to make sure you can load the manifest in the browser and that its content type is correctly set. You can also try renaming it to `manifest.json`.

For Chromium browsers

In DevTools

1. In the left pane, under **Application**, select **Manifest**.
2. Check the fields of the manifest as parsed by the browser.

For Firefox

1. Open the Inspector.
2. Go to the Application tab.
3. Select the Manifest option in the left panel.
4. Check the fields of the manifest as parsed by the browser.

It's not possible to debug web app manifests in Safari, the only option you have to see if it's working is to try to install your PWA on iOS or iPadOS.

Designing your PWA experience

With your PWA now connected to its manifest, it's time to fill out the rest of the fields to define the experience for your users.

Before the web app manifest spec was defined, several browsers, including Safari on iOS/iPadOS and Chrome on Android, supported custom `<meta>` elements to describe the application experience, such as `apple-mobile-web-app-capable`. Do not use these `<meta>` elements today; it's no longer recommended, and may harm the installation experience when the browser can't load the manifest properly; the experience you get as a fallback may be different and unexpected.

Basic fields

The first set of fields represents the core information about your PWA. They are used to build the installed PWA's icon and window and determine how it starts up. They are:

`name` : Full name of your PWA. It will appear along with the icon in the operating system's home screen, launcher, dock, or menu.

`short_name` : Optional, a shorter name of your PWA, used when there is not enough room to display the full value of the `name` field. Keep it under 12 characters to minimize the possibility of truncation.

`icons` : Array of icon objects with `src`, `type`, `sizes`, and optional `purpose` fields, describing what images should represent the PWA.

`start_url` : The URL the PWA should load when the user starts it from the installed icon. An absolute path is recommended, so if your PWA's home page is the root of your site, you could set this to '/' to open it when your app starts. If you don't provide a start URL, the browser can use the URL the PWA was installed from as a start. It can be a deep link, such as the details of a product instead of your home screen.

`display` : One of `fullscreen`, `standalone`, `minimal-ui`, or `browser`, describing how the OS should draw the PWA window. You can read more about the different display modes in the App Design chapter. Most use cases implement `standalone`.

`id` : A string that uniquely identifies this PWA against others that may be hosted on the same origin. If it's not set, the `start_url` will be used as a fallback value. Keep in mind that by changing the `start_url` in the future (such as when changing a query string value) you may be removing the browser's ability to detect that a PWA is already installed.

If you published a PWA before September 2021 read about the best practice for adding an `id` property now in Uniquely identifying PWAs with the web app manifest id property.

Some manifest fields are not supported by every browser. When a browser doesn't support a field, it's just ignored and your manifest is still valid. You can learn more about each field at Web App Manifests.

Icons

Your PWA's icon is its visual identity across your users' devices when installed, so it's important to define at least one. Because the `icons` property is a collection of icon objects, you can define several icons in different formats to provide the best icon experience for your users. Each browser will pick one or more icons based on its needs and the operating system it's installed on, the icons closer to the specifications needed.

While PNGs can include transparency, providing transparent icons may cause unexpected issues on some operating systems. For example, iOS, iPadOS, and Android 8 and later fill in the transparency with a background color out of your control. For better compatibility, use a square non-transparent icon.

If you need to pick only one icon size, it should be 512 by 512 pixels. However, providing more sizes is recommended including 192 by 192, 384 by 384, and 1024 by 1024 pixel-sized images, too.

```
"icons": [
   {
```

```
      "src": "icons/512.png",
      "type": "image/png",
      "sizes": "512x512"
   },
   {
      "src": "icons/1024.png",
      "type": "image/png",
      "sizes": "1024x1024"
   }
]
```

Some resources will suggest you add more icon sizes for Android, such as 48 by 48, 72 by 72, 96 by 96, 144 by 144, 168 by 168, 256 by 256. While you can add as many sizes as you want, these are no longer necessary as they are Android icon sizes for legacy devices; even if you still have some users expecting those resolutions, the browser will resize one of your shipped icons.

If you don't provide an icon or the icons are not in the recommended sizes, on some platforms you won't pass installation criteria. On other platforms, the icon will be automatically generated, for instance from a screenshot of the PWA or by using a generic icon.

Maskable icons

Some operating systems, such as Android, adapt icons to different sizes and shapes. For example, on Android 12, different manufacturers or settings can change the shape of icons from circles to squares to rounded-corner squares. To support these kinds of adaptive icons, you can provide a maskable icon using the `purpose` field.

To do so, provide a square image file that has its main icon contained within a “safe zone”, a circle centered in the icon with a radius of 40 percent of the width of the icon. (See the image below.) Devices that support maskable icons will mask your icon as needed.

The safe area marked as a 40 percent radius centered circle within the square icon

If you don't provide a maskable icon, devices that expect them may shrink your icon and make it into the shape they need with a white background.

Here's an example of a maskable icon rendered in a number of commonly used shapes:

In the following image, if you use the icon at the left as a maskable icon, you will end up with poor results on devices when a shape mask is applied.

An icon that is not suitable for a maskable icon.

This image could be made usable with more padding.

The icon with more padding is suitable for masks.

Maskable icons should be 512 by 512 at least. With one created, you can add it to your `icons` collection to improve the experience for supported devices:

```
"icons": [
   {
      "src": "/icons/512.png",
      "type": "image/png",
      "sizes": "512x512"
   },
   {
      "src": "/icons/1024.png",
      "type": "image/png",
      "sizes": "1024x1024"
   },
   {
      "src": "/icons/512-maskable.png",
      "type": "image/png",
      "sizes": "512x512",
      "purpose": "maskable"
   },
]
```

In most cases, if your maskable icon isn't displaying well, you can improve it by adding more padding. Maskable.app is a free online tool to test and create a maskable version of your icon.

Recommended fields

The next set of fields to include are ones that will improve your user's experience, even though they're not required for installability.

`theme_color` : Default color for the application, sometimes affecting how the OS displays the site (for instance, the window and title bar color on desktop, or the status bar color on mobile devices). This color can be overridden by the HTML `theme-color` `<meta>` element.

`background_color` : Placeholder color to display in the application's background before its stylesheet is loaded. Safari on iOS and iPadOS and most desktop browsers currently ignore this field.

`scope` : Changes the navigation scope of the PWA, allowing you to define what is and isn't displayed within the installed app's window. For example, if you link to a page outside of the scope, it will be rendered in an in-app browser instead of within your PWA window. This will not, however,

change the scope of your service worker.

The next image shows how the `theme_color` field is used for the title bar on a desktop device when you install a PWA.

The same PWA installed on desktop with a different theme color.

If you want to provide different theme colors for dark and light modes, you can override the manifest's value with a meta tag in the HTML using a media query, such as with `<meta name="theme-color" content="red" media="(prefers-color-scheme: dark)">`

When defining colors in the manifest, such as within `theme_color` and `background_color`, you should use CSS named colors, such as `salmon` or `orange`, RGB colors such as `#FF5500`, or color functions without transparency such as `rgb()` or `hsl()`. Check the App design chapter for more information.

Do not use transparency, CSS variables, gradient functions, or color functions with transparency (such as `rgba()`) as they are not supported by most browsers. You will get inconsistent results.

Try it

Splash screens

On some devices, a static image is rendered while your PWA is being loaded to provide immediate feedback to the user.

Android uses the `theme_color`, `background_color`, and `icon` values to generate the splash screen.

When you install a PWA on Android, the device will generate a splash screen with the information that comes from your manifest as seen in the following diagram.

A PWA on Android splash screen taking different values from the manifest.

Safari on iOS and iPadOS, on the other hand, doesn't use the web app manifest to generate splash screens. Instead, they use an image linked from a proprietary `<link>` element similar to how they handle icons. Check the Enhancement chapter for more details.

Extended fields

The next set of fields offers additional information about your PWA. They are all optional.

`lang` : A language tag specifying the primary language of the manifest's values, such as `en` for English, `pt-BR` for Brazilian Portuguese, or `in` for Hindi.

`dir` : The direction to display direction-capable manifest fields (such as `name`, `short_name`, and `description`). Valid values are `auto`, `ltr` (left-to-right), and `rtl` (right-to-left).

`orientation` : Desired orientation for the app once installed. A game may set this to request a landscape-only orientation. Several values are accepted, but if included it's typically `portrait` or `landscape` explicitly.

The Web Application Manifest spec doesn't allow internationalization within the file to localize fields' values and images. To support multilingual PWAs with different web app manifest values you need to implement a custom solution to change those values per language. Watch 5 tips for your PWA for more ideas.

Promotional fields

The fourth set of fields lets you provide promotional information about your PWA, for instance, in install flows, listings, and search results.

`description` : An explanation of what the PWA does.

`screenshots` : Array of screenshot objects with `src`, `type`, and `sizes` (similar to the `icons` object) intended to showcase the PWA. There are no size restrictions.

`categories` : Array of categories the PWA should belong to be used as hints for listings, optionally from the list of known categories]. These values are typically lowercase.

`iarc_rating_id` : The International Age Rating Coalition certification code for the PWA, if you have one. It is intended to be used to determine which ages your PWA is appropriate for.

Promotional fields have been moved to their own spec: Web App Manifest - Application Information. Check that spec for the latest additions to these fields.

You can see these promotional fields in action today. On Android, for example, if your PWA is installable and you provide values for at least the `description` and `screenshots` fields, the installation dialog experience transforms from a simple "Add to the home screen" info bar, to a richer installation dialog similar to the one from an app store.

On Android, you can get a nicer installation UI if you provide values for the promotional fields, as you can see in the next video

See these promotional fields in action:

Capabilities Fields

Finally, there are a number of fields related to different capabilities that your PWA can use in supported browsers, such as the `shortcuts`, `share_target`, `display_overrides` fields as we cover in the

Capabilities chapter. There are also fields, like `related_apps` and `prefer_related_apps` (see the Detection chapter for more information), to connect your PWA to installed apps, often from an app store.

Many new fields may appear in the future while browsers add more capabilities to Progressive Web Apps.

Resources

- Add a Web App Manifest
- Adaptive icon support in PWAs with maskable icons
- Richer PWA installation UI
- MDN: Web App Manifest

□□□ #3
□ □□□ □□□ 2026-10-31 22:56:08 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:32 UTC