App design

- The window
- The icon
- Theming your app
- Display modes
  - Fullscreen experience
  - Standalone experience
  - Minimal user interface
- Optimizing design for desktop
- CSS best practices
  - Media Queries
- The app experience
  - User selection
  - Accent color
  - System fonts
  - Pull to refresh
  - Safe areas
- Resources

This chapter focuses on some critical aspects of rendering content outside of the browser tab.

The window

Different operating systems have different ideas about what an application window is. For example, on iPhones, an application always takes up 100% of the screen. On Android and iPads applications usually run fullscreen but it is possible to share the screen between two apps, however there is only one app instance open at a time. In contrast, on a desktop device, an application can have more than one instance open at a time. It shares the available screen real estate with all other open applications. Each application instance can be resized and positioned anywhere on the screen, even overlapping other applications.

The icon

We recognize apps by their icon. That icon appears when you search for apps, in settings, wherever you launch apps, and where you see running apps.

These include:

- Home screen (iOS, iPadOS, Android).
- App Launcher (macOS, Android).
- Start Menu or App Menu (Windows, ChromeOS, Linux).
- Dock, TaskBar, or Multi-task panels (all operating systems).

When creating the icon for your Progressive Web App make sure its icon is platform-agnostic, as each operating system can render icons and apply different shape masks to them, like the ones in the image below.

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/geo6MZwaWvNDRFJe6cvF.png", alt="PWA icons in different shapes for different platforms.", width="800", height="207" %}

Theming your app

There are a number of ways you can customize app styling in your PWA's, including:

- Theme color: defines the color of the window's title bar on the desktop and the color of the status bar on mobile devices. Using a meta tag, you can have options for different schemes, such as dark or light mode and they will be used based on the user's preference.
- Background color: defines the color of the window before the app and its CSS are loaded.
- Accent color: defines the color of built-in browser components, such as form controls.

  {% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/fW7mXimvH0Djb637xDH8.png", alt="A desktop PWA showing theme and accent colors, and an Android PWA splash screen showing theme and background colors.", width="800", height="526" %}
  A desktop PWA showing theme and accent colors, and an Android PWA splash screen showing theme and background colors.

{% Aside %}
When planning your Progressive Web App user interface, beware that you can't use images, or gradients for your theme, just plain colors. It's best to pick colors without transparency, and use the `rgb` or `hsl` colors, hex codes, or named colors. Theming your app may also affect browser theming even without installing the app, such as when using the `<meta name="theme-color">` element. {% endAside %}

Display modes

You can define what kind of window experience you'd like for your Progressive Web App. There are three options to choose from:

- Fullscreen
- Standalone
- Minimal User Interface

{% Aside %} If a particular display mode isn't supported or otherwise can't be used to display your PWA, it'll fall back to the next available display mode. Fullscreen will fall back to standalone, standalone will fall back to minimal user interface, and minimal user interface will fall back to browser, the default. The browser display mode doesn't show up as its own window, but rather displays your PWA in a tab in the browser it was installed from, like a web bookmark on a home screen. {% endAside %}

Fullscreen experience

A fullscreen experience is suitable for immersive experiences, such as games, VR, or AR experiences. It's currently only available on Android devices, and it hides the status bar and the navigation bar, giving your PWA 100% of the screen for your content.

On desktop and iPadOS, fullscreen PWAs are not supported; however, you can use the Fullscreen API from within your PWA to display your app fullscreen at a user's request.

Standalone experience

The most common option for a Progressive Web App, standalone mode displays your PWA in an OS-standard window without any browser navigation UI. The window may also include a browser-controlled menu where the user can:

- Copy the current URL.
- See, apply, or disable browser extensions.
- See and change permissions.
- Check current origin and the SSL certificate.

The title bar may also show permissions and hardware usage replacing the omnibox or URL bar when the PWA renders in the tab.

> {% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/0496WcZmxpNXE35jOGEw.png", alt="A PWA installed with Microsoft Edge on desktop showing its menu.", width="800",

height="459" %}
{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/2RmIdDEe4gm99lTzwLCa.png", alt="A PWA installed with Google Chrome on desktop showing the drop-down menu and the plugins-icon.", width="800", height="640" %}

The images above show how a PWA displays in standalone mode on the desktop in Microsoft Edge, and Chrome.

On mobile devices, a standalone PWA experience will create a standard screen that keeps the status bar visible, so the user can still see notifications, time, and battery level. It often does not have any browser-controlled menu like desktop standalone experiences may include.

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/sNa6aVY2ZayRqPZhpfIl.png", alt="An iOS device rendering a standalone app.", width="800", height="476" %}

Some browsers on Android create a fixed and silent notification while the PWA is in the foreground that lets the user copy the current URL or other options.

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/iE1jwkZq1VWMJC8RZxJs.png", alt="An Android notification rendered by Chrome showing some actions over the current active PWA.", width="730", height="428" %}

Minimal user interface

This mode is available for Progressive Web Apps on Android and desktop operating systems. When you use it, the browser rendering your PWA will show a minimal user interface to help the user navigate within the application.

On Android, you will get a title bar that renders the current `<title>` element and the origin with a small drop-down menu available. On desktop, you will get a set of buttons in the title bar to help with navigation, including a back button and a control that swaps between a stop and a reload action, based on the current loading status.

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/nzzjXh6YK5GzP6K2Sfb5.png", alt="A desktop minimal-ui on Microsoft Edge with back and reload buttons", width="800", height="399" %}

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/wu1S51rlQpgBcDkJgEvG.png", alt="On Android, browsers use a read-only themed navigation bar for minimal-ui, here Firefox and Chrome", width="800", height="249" %}

{% Aside 'warning' %} Safari on iOS and iPadOS doesn't support Minimal UI mode for PWAs. Instead, a browser shortcut is used as a fallback. The user can add an icon to the home screen, but that icon will point to Safari as a bookmark, and they won't have an app experience out of the browser tab. {% endAside %}

Optimizing design for desktop

When you design a Progressive Web App to work on a desktop, you need to think about the endless possibilities for window size as compared to being in the browser's tab or as an app in a mobile operating system.

In Chapter 3, we mentioned mini-mode: a desktop app can be as small as 200 by 100 pixels. This window will use the `<title>` element's content in your HTML as the title of the window. That content is also rendered when you **alt-tab**"** between apps and in other places.

Pay attention to your HTML's `<title>` element and rethink how you use it. The `<title>` isn't just for SEO or to render just the first characters in a browser's tab; it will be part of the user experience of your standalone desktop window.

{% Aside %} If you are developing your app using a client-side library, such as React or Vue, don't forget to change the `<title>` to some practical value when moving between sections or routes of your app. It's time to reimagine the `<title>`! {% endAside %}

CSS best practices

All your experience with CSS layout, design, and animation is valid when your content is rendered in its standalone experience. However, there are a couple of tips and tricks to making the experience better for a standalone window.

{% Aside 'gotchas' %} Remember that CSS only defines styles for the content of the PWA; later in this course, we will see how to specify style declarations for the window around your content, such as properties in the web app manifest. {% endAside %}

Media Queries

The first media query that you can take advantage of in your PWA is the `display-mode` property accepting the values `browser`, `standalone`, `minimal-ui`, or `fullscreen`.

This media query applies different styles to each mode. For example, you could render an installation invitation only when in browser mode, or render one particular piece of information only when the user uses your app from the system icon. This might include adding a back button to use when your app launches in standalone mode.

```
/* It targets only the app used within the browser */
@media (display-mode: browser) {
}
/* It targets only the app used with a system icon in standalone mode */
@media (display-mode: standalone) {
}
/* It targets only the app used with a system icon in all mode */
@media (display-mode: standalone), (display-mode: fullscreen), (display-mode: minimal-ui) {
}
```

{% Aside %} Users expect apps to integrate with the platform, so consider offering dark and light modes using `prefers-colors-scheme` and honoring `prefers-reduced-motion` media queries to disable or reduce transitions and animations. {% endAside %}

The app experience

When the users are using an installed PWA, they expect app behavior. While it's not simple to define what it means, the default web behavior doesn't give the best experience in some situations.

User selection

Content is generally selectable with a mouse or pointer, or a press and hold touch gesture. While helpful for content, it doesn't provide the best experience for navigation items, menus, and buttons within your PWA.

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/F9HCkRrmPyGFjlpTJhNz.png", alt="A calculator PWA where you can select every interactive button, such as the numbers.", width="800", height="739" %}

Therefore, it's a good idea to disable user selection on these elements using `user-select: none` and it's `-webkit-` prefix version:

```
.unselectable {
   user-select: none;
}
```

{% Aside 'gotchas' %} User selection is a vital feature of every web app for usability and accessibility. Do not disable user selection on all content, only on UI elements when not doing so creates a poor or unexpected experience. {% endAside %}

Accent color

In your PWA, you can define a color to match your brand within HTML form controls using the property `accent-color`.

System fonts

If you want an element, like dialogs or messages, to match the user's default platform font, you can use the following font family:

```
selector {
  font-family: -apple-system, BlinkMacSystemFont,
    "Segoe UI", system-ui, Roboto, Oxygen-Sans, Ubuntu, Cantarell,
    "Helvetica Neue", sans-serif;
}
```

Pull to refresh

Modern mobile browsers, such as Google Chrome and Safari, have a feature that refreshes the page when it is pulled down. On some browsers, such as Chrome on Android, that behavior is also enabled on standalone PWAs.

You may want to disable this action. For example when providing your own gesture management or refresh action, or if there is a possibility that your user will trigger the action unintentionally.

It's possible to disable this behavior by using `overscroll-behavior-y: contain`:.

```
  body {
    overscroll-behavior-y: contain;
  }
```

Safe areas

Some devices do not have unobstructed rectangular screens; instead, their screen may be a different shape, like a circle, or have portions of the screen that can't be used, like the iPhone 13's notch. In these cases, some browsers will expose environment variables with safe areas that can display content.

{% Img src="image/RK2djpBgopg9kzCyJbUSjhEGmnw1/hKWqUXIbvvH05WHQEccm.png", alt="At top, a notch-based device in landscape with a standard viewport showing unrendered areas at the sides; at bottom, a device with full viewport access thanks to viewport-fit=cover.", width="800", height="1006" %}

If you want full access to the screen, even the invisible area, to render your color or image, include `viewport-fit=cover` in the content of your `<meta name="viewport">` tag. Then use the `safe-area-inset-*` environment variables to extend your content safely into those areas.

{% Aside 'caution' %} When you apply `viewport-fit=cover`, remember that you can now render pixels behind rounded corners and notches, so you should always use safe margins or paddings for the critical content and interactive elements. {% endAside %}

Resources

- CSS display-mode media query
- Take control of your scroll: customizing pull-to-refresh and overflow effects
- prefers-color-scheme: Hello darkness, my old friend
- prefers-reduced-motion: Sometimes less movement is more
- env() CSS function
- CSS Accent Color

□□□ #2
□ □□□ □□□ 2026-10-31 22:54:49 UTC
□ □□□ □□□ 2026-10-31 22:54:51 UTC