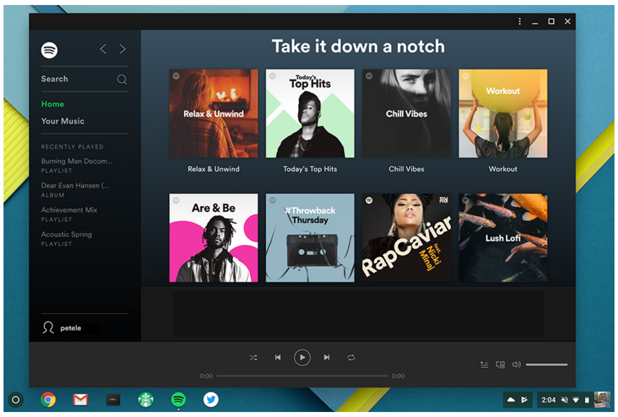
Progressive Web Apps on the Desktop

By Pete LePage
Pete is a Developer Advocate

Success: Support for Desktop Progressive Web Apps is supported on Chrome OS 67, which was released in May, 2018.

Dogfood: Work is currently under way to support Mac and Windows. You can test support by enabling the `#enable-desktop-pwas` flag.

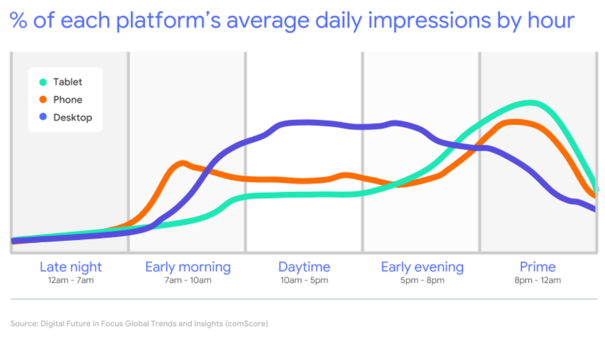


Desktop progressive web apps can be 'installed' on the user's device much like native apps. They're **fast**. Feel **integrated** because they launched in the same way as other apps, and run in an app window, without an address bar or tabs. They're **reliable** because service workers can cache all of the assets they need to run. And they create an **engaging** experience for users.

Check out my Google I/O talk PWAs: building bridges to mobile, desktop, and native, for more about what's new with Progressive Web Apps and some important best practices you should be following, or jump right to the section on desktop PWAs

Desktop usage is important

Mobile has driven a lot of the evolution of Progressive Web Apps. But while the growth of mobile has been so strong, desktop usage is still growing. Mobile phone use peaks in the morning and evening, and tablet also has significantly higher use in the evening. Desktop usage is more evenly distributed throughout the day than mobile usage. It has significant use during the day when most people are at work and at their desks.

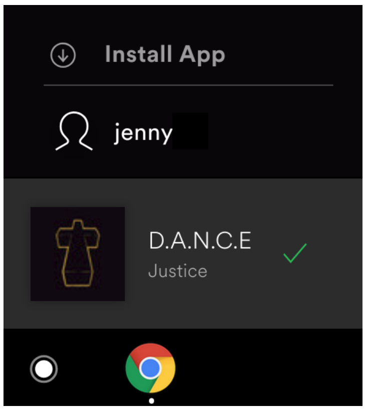


Having that 'installed', native feel, is important to users, it gives them the confidence that the app will be fast, integrated, reliable and engaging. Desktop Progressive Web Apps can be launched from the same place as other desktop apps, but that they run in an app window - so they look and feel like other apps on the desktop.

Getting started

Getting started isn't any different than what you're already doing today; it's not like this is a whole new class of apps. All of the work you've done for your existing Progressive Web App still applies. Service workers make it works fast, and reliably; Web Push and Notifications keep users updated, and it can be 'installed' with the add to home screen prompt. The only real difference is that instead of running in a browser tab, it's running in an app window.

Add to home screen



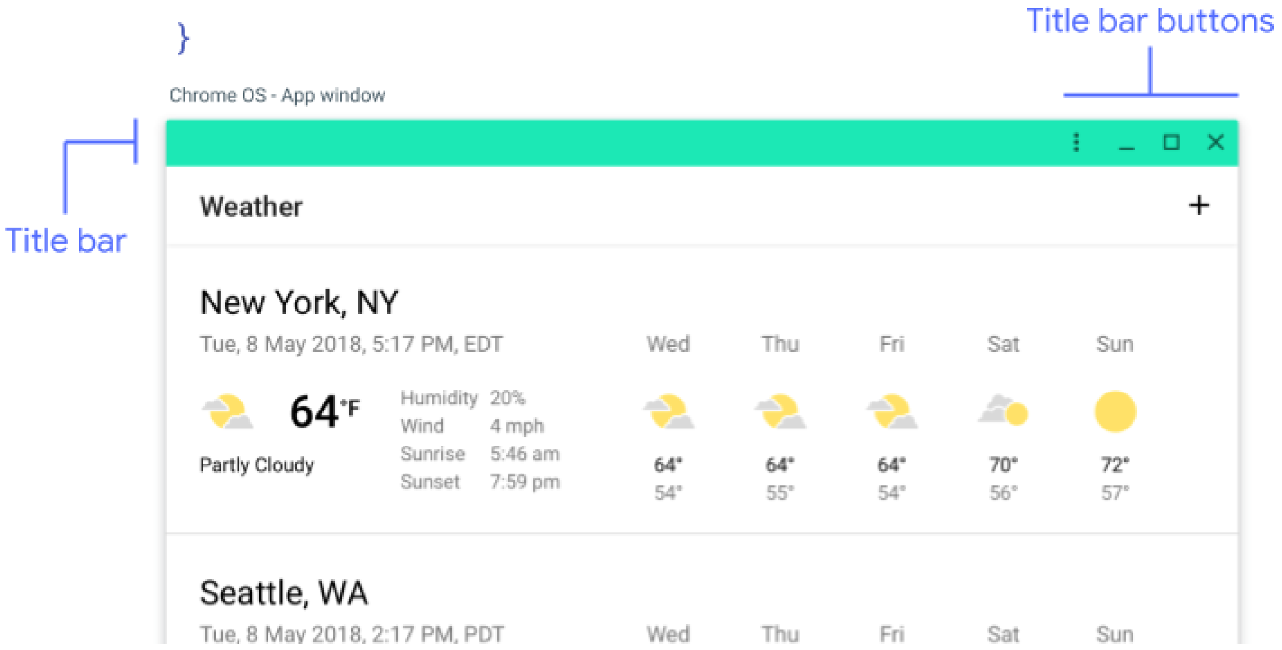
If the add to home screen criteria are met, Chrome will fire a `beforeinstallprompt` event. In the event handler, save the event, and update your user interface to indicate to the user that they can add your app to the home screen. For example, Spotify's desktop Progressive Web App, adds an 'Install App' button, just above the users profile name.

See Add to Home Screen for more information about how to handle the event, update the UI and show the add to home screen prompt.

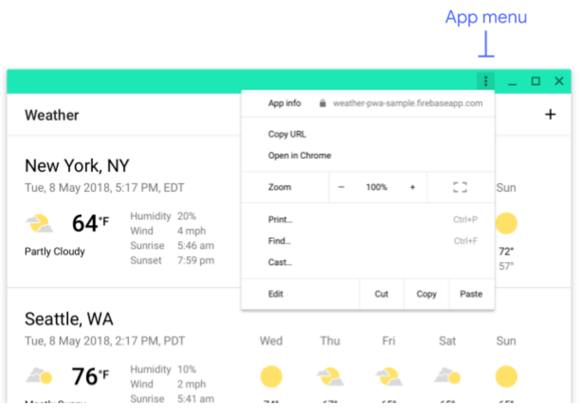
The app window

With an app window, there are no tabs or address bar, it's just your app. It's optimized to support the needs of apps, with more flexible window organization and manipulation compared to browser tabs. App windows make it easy to uni- task with the window in full screen, or multi-task with multiple windows open. App windows also make it really easy to switch between apps using an app switcher or a keyboard shortcut such as alt-tab.

```
{...
  "theme_color": "#1CE8B5",
}
```



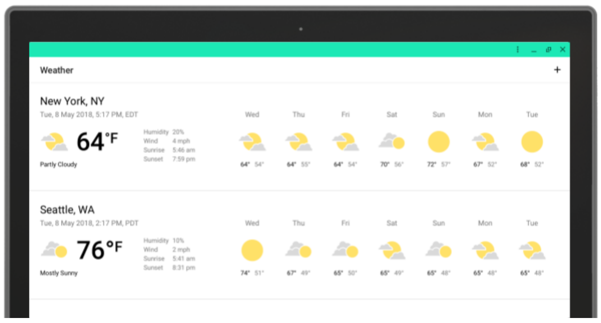
As you'd expect, the app window has the standard title bar icons to minimize, maximize and close the window. On Chrome OS, the title bar is also themed, based on the `theme_color` defined in the web app manifest. And your app should be designed to take up the full width of the window.



Within the app window, there's also the app menu (the button with the three dots), that gives you access to information about the app, makes it easy to access the URL, print the page, change the page zoom, or open the app in your browser.

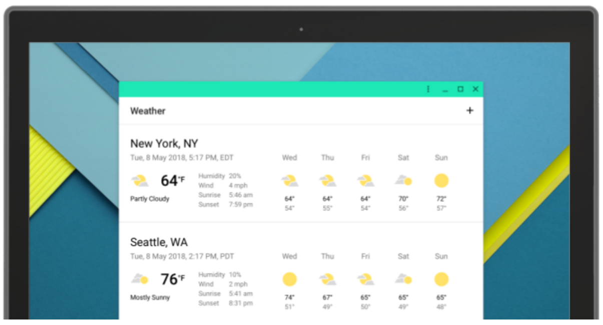
Design considerations

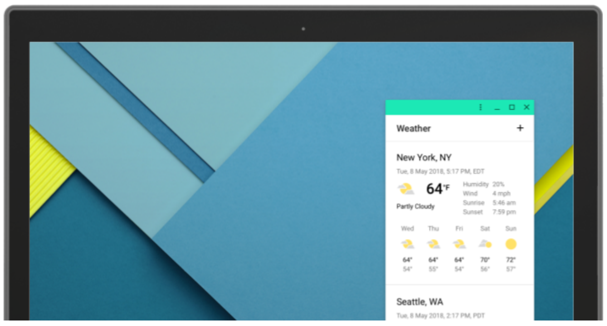
There are some unique design considerations you need to take into account when building Desktop Progressive Web Apps, things that don't necessarily apply to Progressive Web Apps on mobile devices.

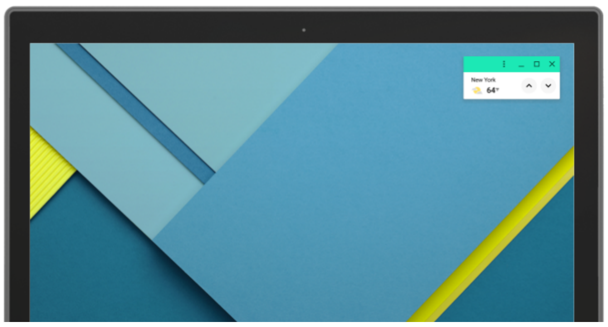


Apps on the desktop have access to significantly larger screen real-estate. Don't just pad your content with extra margin, but use that additional space by creating new breakpoints for wider screens. Some applications really benefit from that wider view.

When thinking about your break-points, think about how users will use your app and how they may resize it. In a weather app, a large window might show a 7 day forecast, then, as the window gets smaller, instead of shrinking everything down, it might show a 5 day forecast. As it continues to get smaller, content might shuffle around, and it's been optimized for the smaller display.







For some apps, a mini-mode might be really helpful. This weather app shows only the current conditions. A music player might only show me the current song and the buttons to change to the next song.

You can take this idea of responsive design to the next level to support convertibles like the Pixelbook or the Surface. When switched to tablet mode, these devices make the active window full screen, and depending on how the user holds the device, may be either landscape or portrait.

Focus on getting responsive design right - and that's what matters here. Whether the user has resized the window, or the device has done so because it's switched to tablet mode, responsive design is critical to a successful desktop progressive web app.

The app window on desktop opens up so many new possibilities. Work with your designer and take a responsive approach that adds new breakpoints for larger screens, supports landscape or portrait views, works when fullscreen - or not, and works nicely with virtual keyboards.

What's next?

We're already working on support for Mac and Windows. For all of these platforms, we're looking at:

- Adding support for keyboard shortcuts, so you can provide your own functionality.
- Badging for the launch icon, so you can let the user know about important events that you don't want to display a full notification for.
- And link capturing - opening the installed PWA when the user clicks on a link handled by that app.

Learn more

Check out my Google I/O talk, **PWAs: building bridges to mobile, desktop, and native**, it covers everything from Desktop PWAs to, upcoming changes to add to home screen prompts, and more.