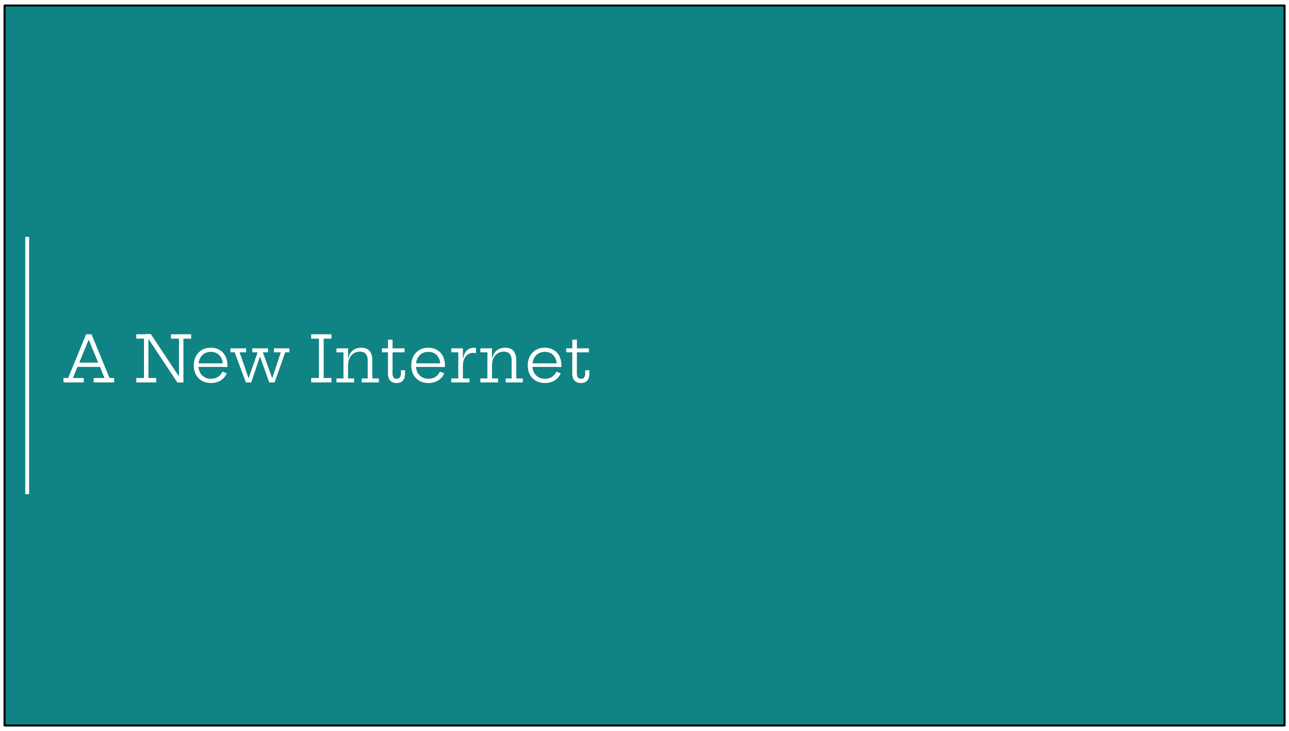
Progressive Web Apps

Past, Present, Future

Sam Richard
@snugug

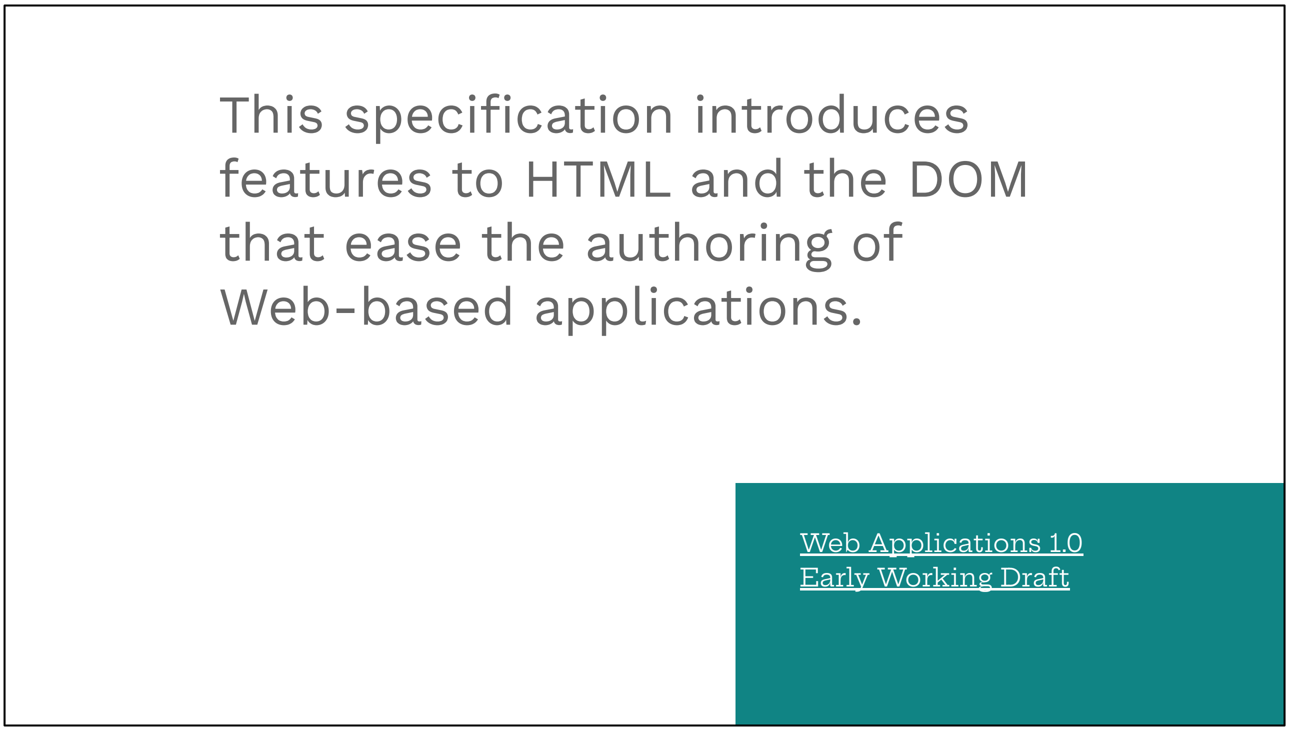
- Sam Richard
- Developer Advocate for Chrome OS, Progressive Web Apps and Web Capabilities at Google
- @Snugug



A New Internet

- The story of PWAs starts with the story of HTML5

- The first working draft of HTML5, then known as Web Applications 1.0, was released in on September 1, 2005.
  - For context, that’s almost 18 months before the first iPhone was released
  - Over 3 years before Android was released
  - Over 4 years before the term Mobile First was coined
  - And over four and a half years before the term Responsive Web Design was coined
- Before all of that, there was this draft, this thought, of what the web could be. The first sentence of its abstract sums up its intention:



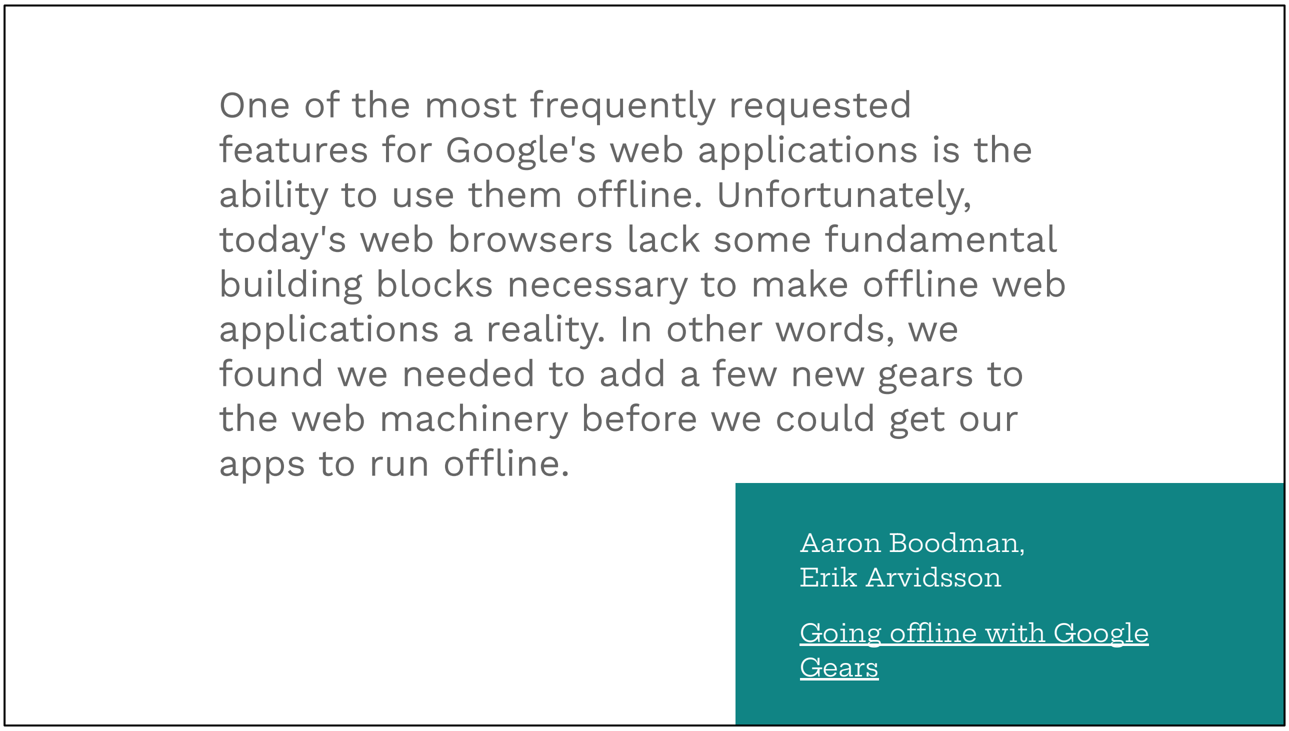
This specification introduces features to HTML and the DOM that ease the authoring of Web-based applications.

Web Applications 1.0
Early Working Draft

- (read quoted text)
- *Applications*. That was the vision for HTML5. That vision, put forth September 1, 2005, that the web could, and should, be a place for *applications*, not just documents, was a powerful one. A vision we're still trying to fulfill.
- But this was 2005. In a few short years, the world the web lived in was about to change out from everyone's feet.
    - A true paradigm shift, if ever there was one, moving from desktop to mobile, was about to unfold.
    - But this idea, that the web could be a place for applications, proved strong enough to withstand the shifting sands of computing.
    - Even more than that, it would show the *need* for the web to make this shift. But that's getting ahead of the story.

HTML5 Offline Webapps

- The Web, up until the introduction of HTML5, was designed to be used online. A series of connected documents.
- With applications as a major driving factor for HTML5, there became a need for web applications to work offline



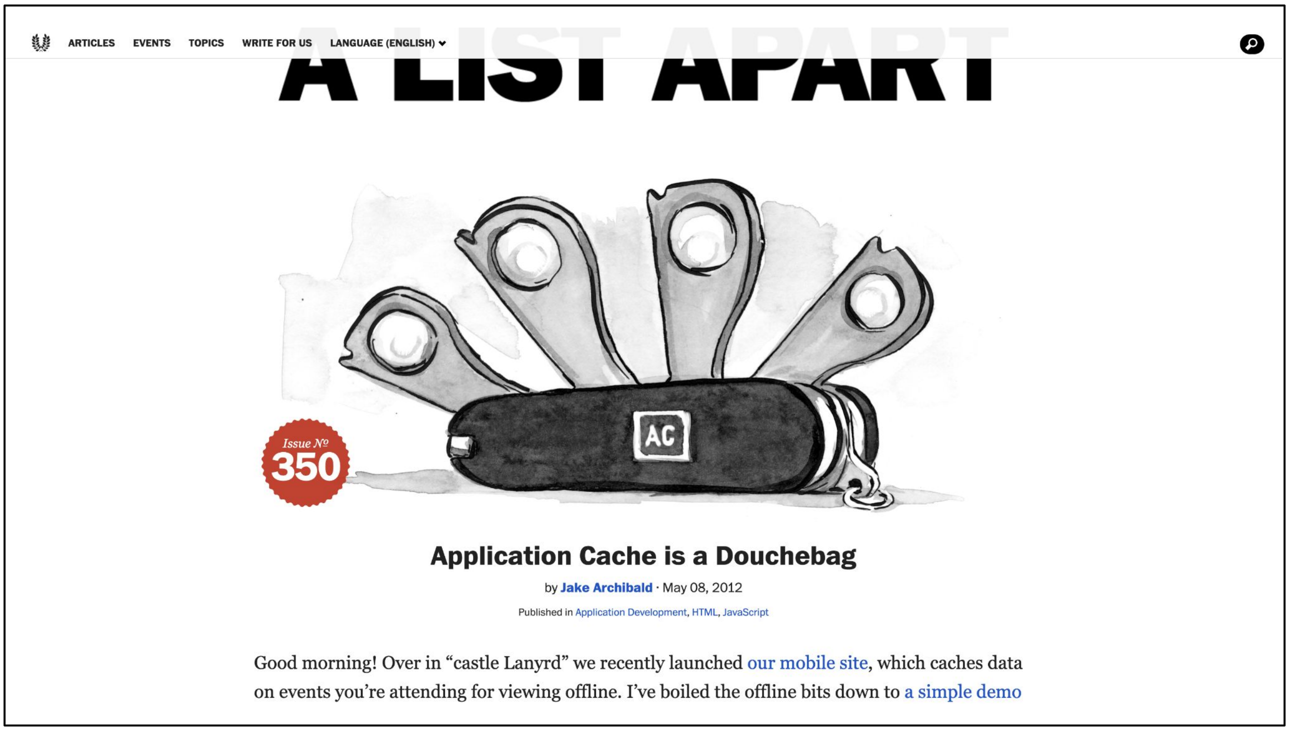
One of the most frequently requested features for Google's web applications is the ability to use them offline. Unfortunately, today's web browsers lack some fundamental building blocks necessary to make offline web applications a reality. In other words, we found we needed to add a few new gears to the web machinery before we could get our apps to run offline.

Aaron Boodman,
Erik Arvidsson

Going offline with Google Gears

- In May of 2007, Google introduced an open source browser extension, Google Gears, designed to help make web apps work offline.
- (read quoted text)
- While not a web standard itself, it was built on top of web standards, adding "just enough AJAX to make current web applications work offline", with the goal of evolving it into a set of web standards.
- Gears included a number of components, that may sound familiar to you today:
    - An SQLLite powered database
    - A worker pool for parallel JavaScript execution
    - A local server that could cache and serve resources
    - A way to create desktop shortcuts
    - And a geolocation API
- While Gears would be officially deprecated in March of 2011 in favor of web standards that had finally caught up, its influence is hard to miss.

- The earliest draft of the HTML5 spec I can find that mentions Offline Web Applications comes from October 2007. A subsequent spec draft from May 2008, talks about a number of capabilities that HTML5 provides to help applications work offline:
    - Client-side SQL
    - Online and offline events
    - localStorage
    - And, finally, “offline application caching APIs”, later known as ApplicationCache or, simply, AppCache
- AppCache was a big deal.
    - For the first time, there was a standards-based way of telling a web browser that certain assets, your CSS, your images, your HTML, should be made available to a user when they’re offline.
- AppCache was entirely declarative.
    - You created a cache manifest file, put in the specific files you wanted the browser to cache, and it, through a series of built-in rules, would.
- The first two browsers that would get full AppCache support were Safari 4 and Firefox 3.5, both released June of 2009.
    - Android Browser 2.1 would get it October of that year, then Chrome 4 January of 2010, then Safari on iOS 3 months later.
    - It wouldn’t land in Opera until mid-2011 or Internet Explorer until the end of 2012.
- AppCache was great in theory, but In practice, it had a lot of gotchas.



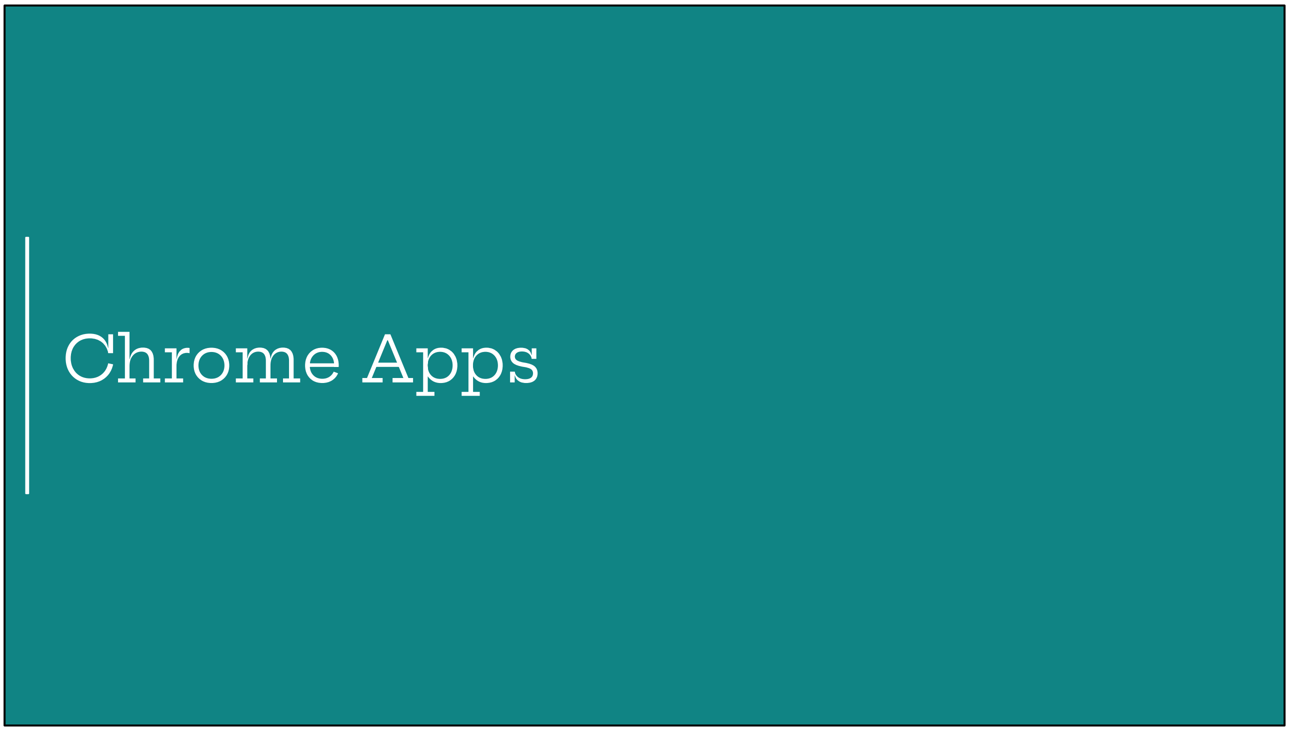
Application Cache is a Douchebag

by Jake Archibald · May 08, 2012

Published in Application Development, HTML, JavaScript

Good morning! Over in "castle Lanyrd" we recently launched our mobile site, which caches data on events you're attending for viewing offline. I've boiled the offline bits down to a simple demo

- Jake Archibald described why it was so hard to work with in his A List Apart article [Application Cache is a Douchebag](). A few key, *painful*, ones mentioned include:
  - Files cached with AppCache would always come from the AppCache, even if the user was online
  - The AppCache would only update if the content of the manifest file had changed, not if the content of the things cached changed
  - Non-cached resources wouldn't load on cached pages, even when online, *by design*
- Replacing AppCache was the start of a new chapter towards what we know as PWAs, but before we get there, we need to take a detour to what web apps looked like on desktops at the time.



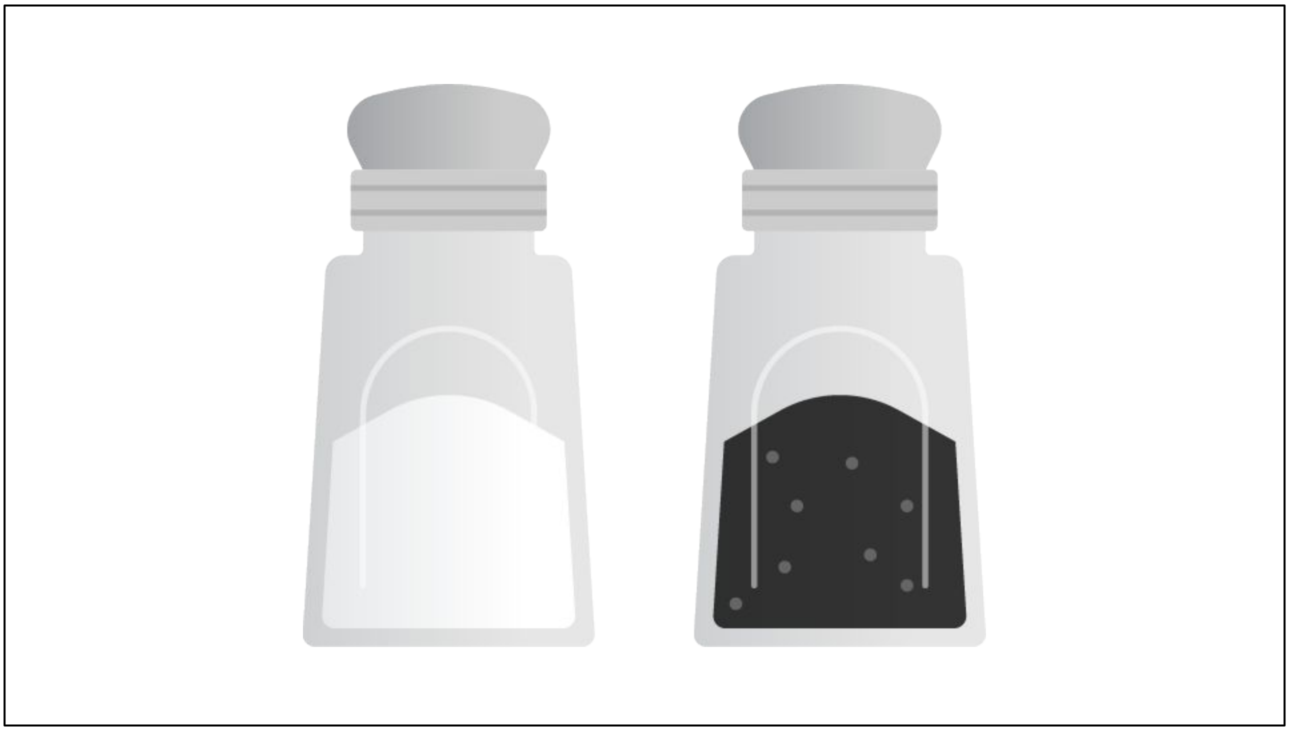
Chrome Apps

- From conversations with members of the Chrome team who were there at the time, what we today know as Progressive Web Apps started life back in 2010.
- At the time, there were two competing schools of thought for how to make the web a better app platform:
  - One school of thought was to work with the standards process and improve the core platform
  - The other school of thought was that the standards process wasn't fast enough for the features they wanted to deliver.

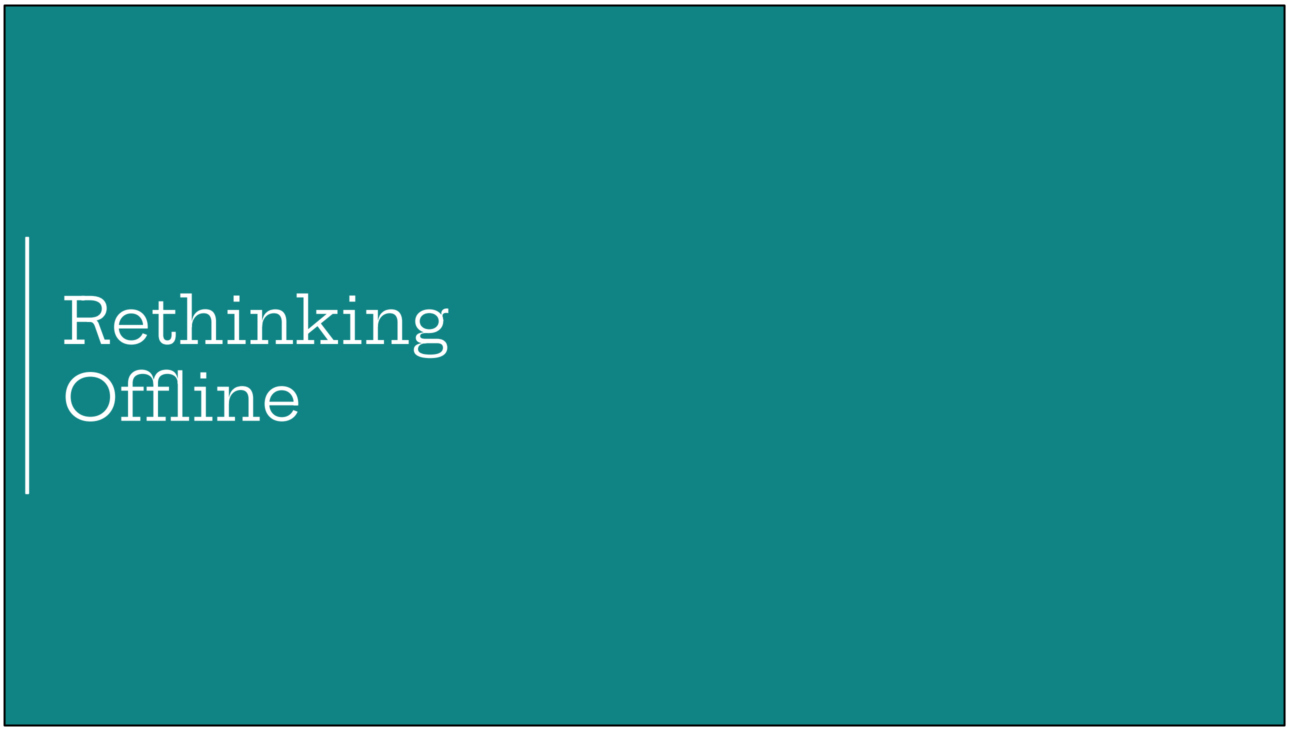
ES6

- The group that wanted to work through the standards process, led by Alex Russell, Dimitri Glazkov, and Alex Komoroske, went on to propose or help drive web standards including
    - Mutation Observers
    - HTML Templates
    - Object.observe
    - Web Components
    - ES Modules
    - ES Classes
    - CSS Variables
    - and Web Animations, to name a few

- The group that wanted to work faster than the platform created Chrome Apps
- Launched December of 2010, the goal of Chrome Apps is a goal you've likely heard from any number of similar solutions, from Electron to Cordova to even Progressive Web Apps today: build fully featured applications that just happen to be built with web technology.
- Chrome Apps existed as either a hosted app, which ran inside of Chrome, or as a packaged app that may sound familiar to you PWA developers out there:
  - Packaged apps could be launched from outside of Chrome. They got their own app windows and could generally behave like other apps
  - They were offline by default. Resources required to run offline came with the app, and it included APIs to make pages resilient under poor network conditions
  - New APIs were added to get low-level hardware access, for instance to USB and Bluetooth, and new shared data APIs to let you more easily inter-operate with other applications on the system.

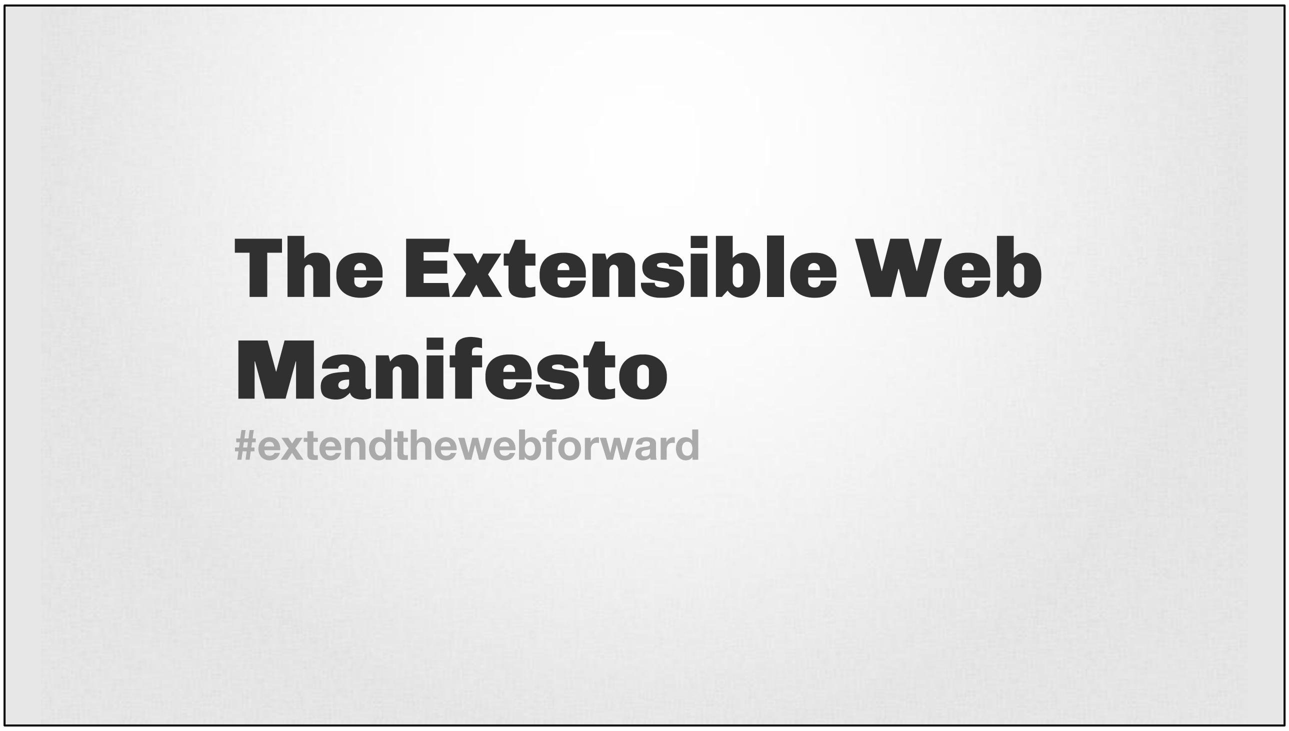


- Less than a year after the release of Chrome Apps, Google Native Client, or NaCl, was released with Chrome 14, September of 2011.
- NaCl, the architecture-independent version Portable Native Client (PNaCl), and Pepper (PPAPI) for creating NaCl modules, allowed Chrome Apps, and web apps, to run C and C++ code in the browser.
  - NaCl was critical to many Chrome Apps by allowing them to run at desktop app speeds and with desktop app capabilities that weren't safe to expose to the general web.
  - Chrome Remote Desktop, for instance, an app to allow Chrome OS users to access their PCs or Macs, used it for handling different transportation protocols and multimedia.
- Today, of course, Chrome Apps are deprecated and NaCl has been replaced with WebAssembly and other web standards initiatives. But, it shouldn't take much squinting to see how concepts that are fundamental to Progressive Web Apps emerged from this period of time.
- Back in the Web Standards world, work was being done to rethink offline apps for the web.



Rethinking Offline

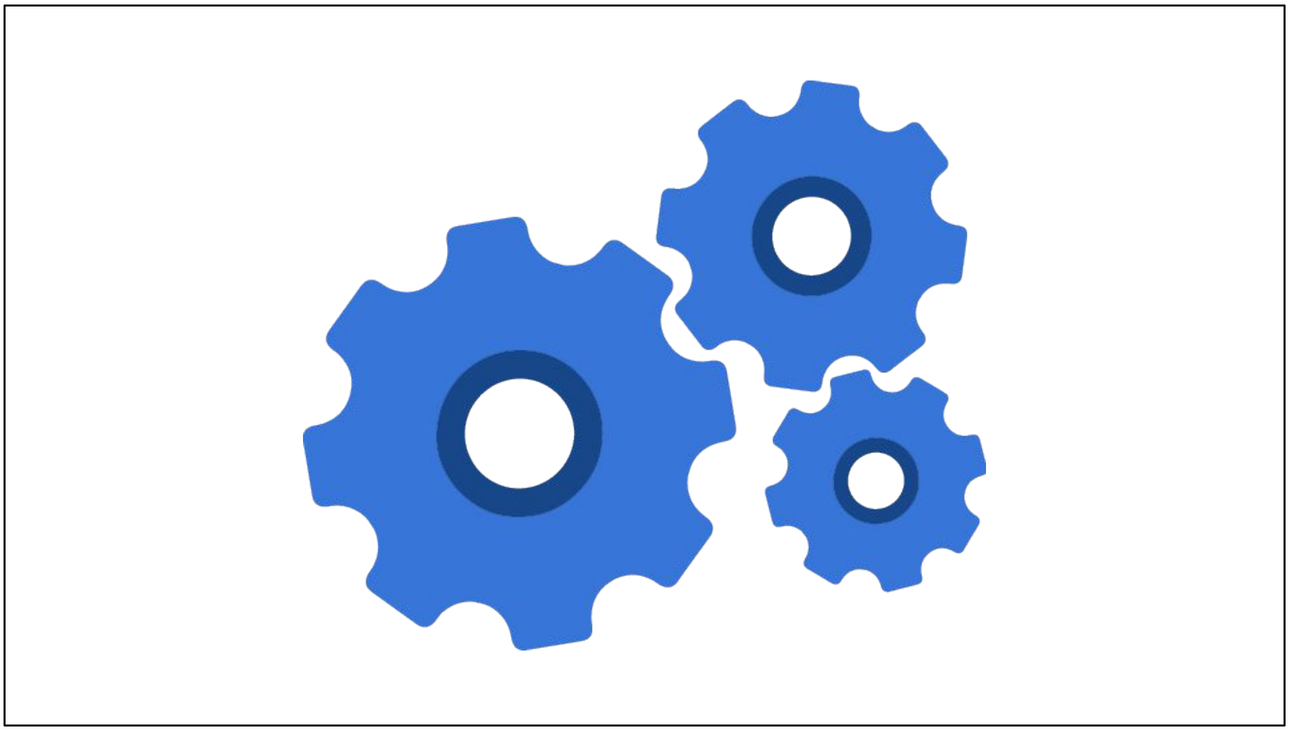
- In August of 2012, a group of developers and browser vendors came together to talk about AppCache, first in London, hosted by FT Labs, then in Mountain View, hosted by Mozilla
  - The goal: talk about practical experiences with sites that developers were presently building.
  - Facebook, Gmail, the Financial Times and the Economist, Twitter, and Microsoft Outlook were amongst the sites discussed
- The conclusion? Not quite tear it all down, but pretty close to it.



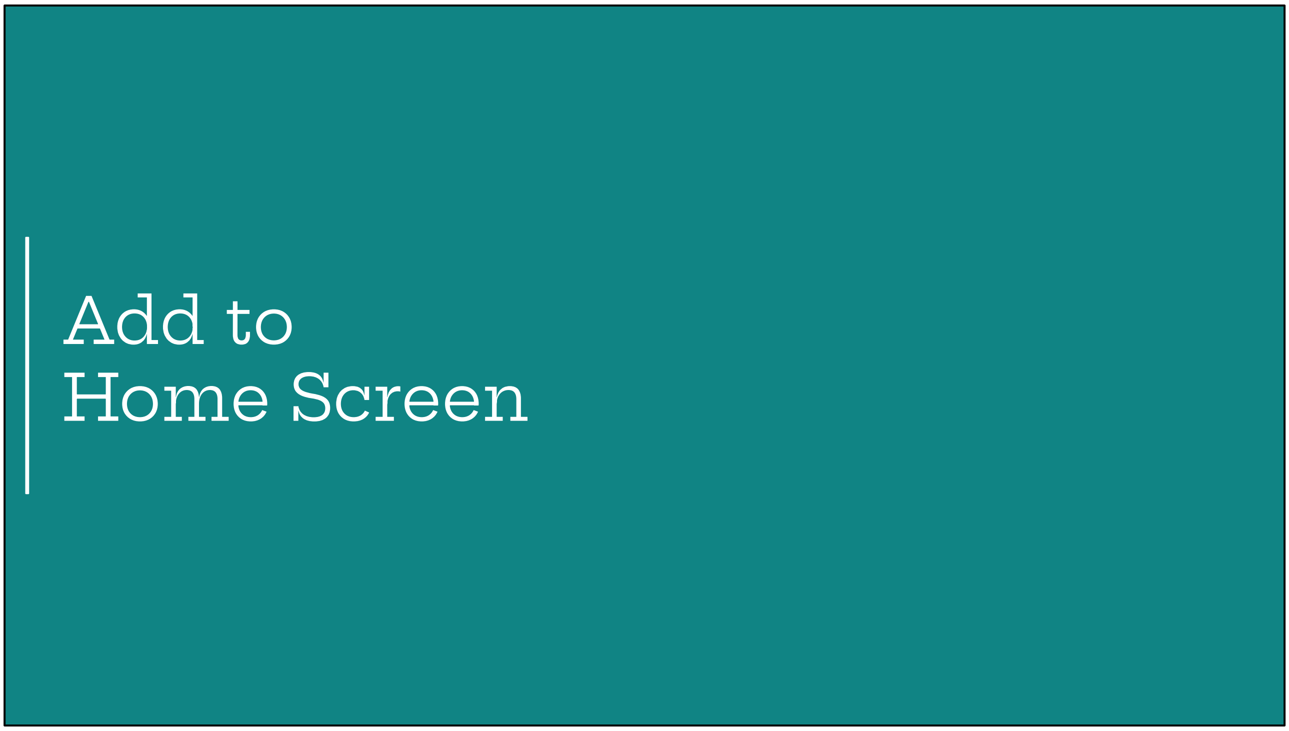
The Extensible Web Manifesto

#extendthewebforward

- In June of 2013, the Extensible Web Manifesto was published.
- Remember that group I mentioned that wanted to improve the web through the standards process? That went on to champion things like ES Modules and Web Components? One of their leads, Alex Russell, at the time a member of TC39, the standards body for JavaScript, the W3C TAG that stewards Web architecture, and the Chrome team would help write the Extensile Web Manifesto. That previous work taught him many lessons that were solidified within it.
- The authors of the Manifesto had seen that, with high-level abstractions, users either needed to buy into the entire thing wholesale, or replace it entirely with an alternative. They believed that users should instead be able to be able to take control **only at the points they needed to** to accomplish the job they wanted.
- To this desire, the manifest offers four design principles for an extensible web:
  - Focus on adding ***new low-level capabilities*** that are secure and efficient
  - Expose low-level capabilities that ***explain existing features***, allowing authors to understand and replicate them
  - Create a ***virtuous cycle*** between standards and developers by developing, describing, and testing new high-level features in JavaScript and allowing developers to iterate on them before they become standardized.
  - ***Prioritize efforts*** that follow these recommendations and deprioritize and refocus those which do not.

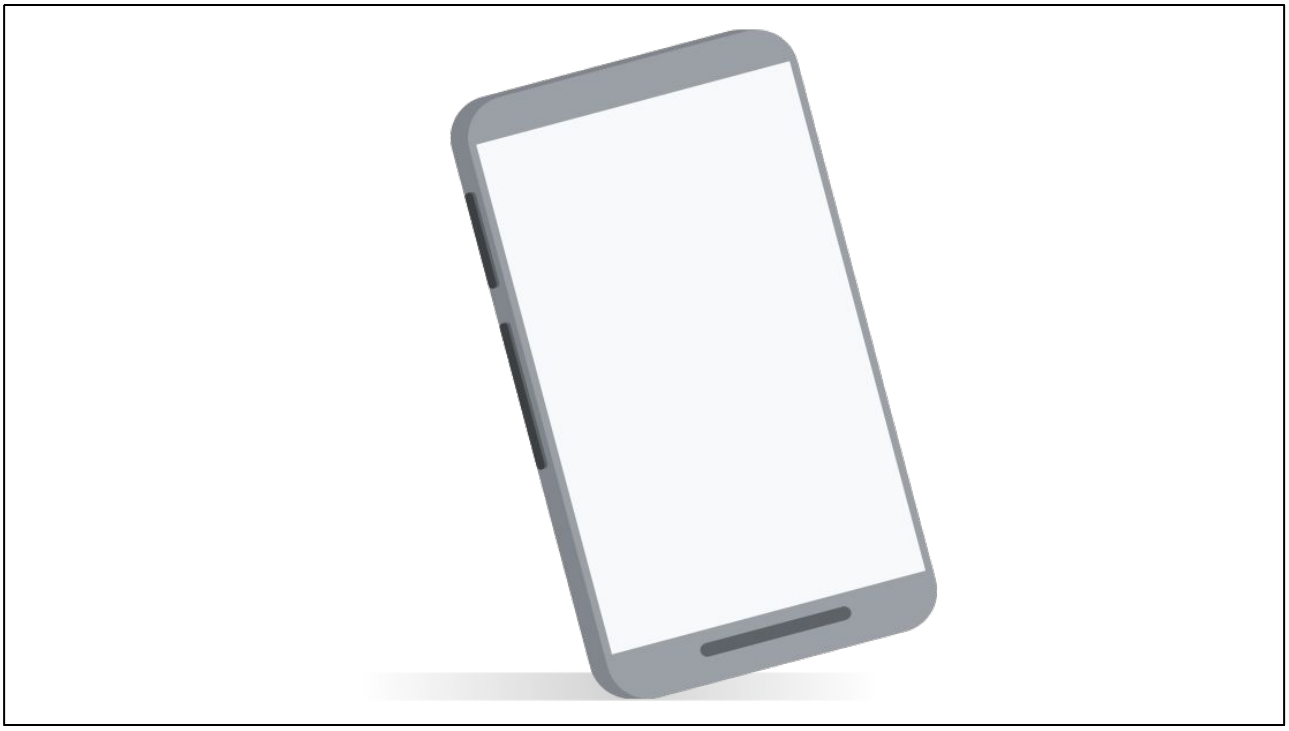


- The first commits for what would become the Service Worker spec were laid down by Alex Russell in February 2013
- By the time of the first public working draft was published on May 8, 2014, the Service Worker spec was shining example of the Extensible Web Manifesto in practice:
    - Where AppCache provided a high-level declarative syntax for caching then had its own rules for deciding what to serve, Service Workers and Cache Storage were low-level APIs that allowed users to script what should happen with JavaScript
    - Using both Service Workers and the Cache Storage API, users could both describe, and recreate, how the browser handled caching *and* how AppCache handled caching.
- Chrome 45, released September 2015, was the first browser with full support for both the Cache Storage API and Service Workers.
    - Firefox 44, released January 2016, would be next, followed by Samsung Internet in April of the same year.
    - Safari 11.1 would get them March of 2018, and finally Microsoft Edge would support them April of 2018
- Some 13 years after it was first described, AppCache has officially been deprecated and removed from browsers, officially replaced by Service Workers, starting with Chrome and Edge 84 released July 2020, followed by Opera 72 and Firefox in October and November.
- Service Workers, though, are only part of the Progressive Web App story. Getting onto the home screen, required something else.

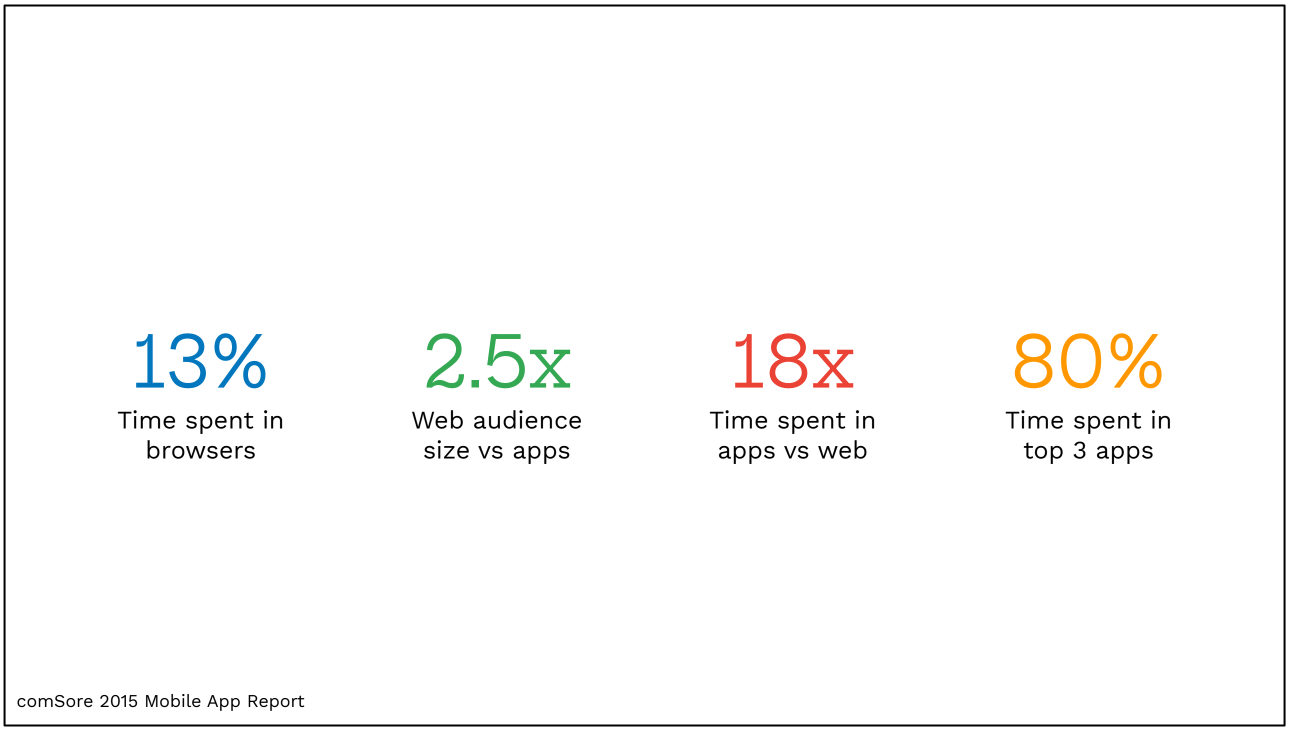


Add to Home Screen

- When the iPhone originally launched in 2007, it didn't have the App Store.
- The original iPhone came with standard apps, Safari 3.1, and the ability to make special website bookmarks, known as "Web Clips", which would add a website to the user's home screen.
- Then, on July 10, 2008, just over 2 months before *Android* was launched, Apple released the App Store, and fundamentally changed how people thought about software; where to find it, how to buy it, what it should cost, and what their relationship with it was.



- As it turned out, software development, especially on the web, was going through not one but *two* paradigm shifts: the move from desktop to mobile, and the move of app discovery from search engines to app stores.
- The web, though, had no app model at the point in time when conversations about software development centered on apps
- The result: uncertainty that the web would survive the tectonic shifts happening. By the time the web had an answer, it had seemingly already lost.



13%
Time spent in browsers

2.5x
Web audience size vs apps

18x
Time spent in apps vs web

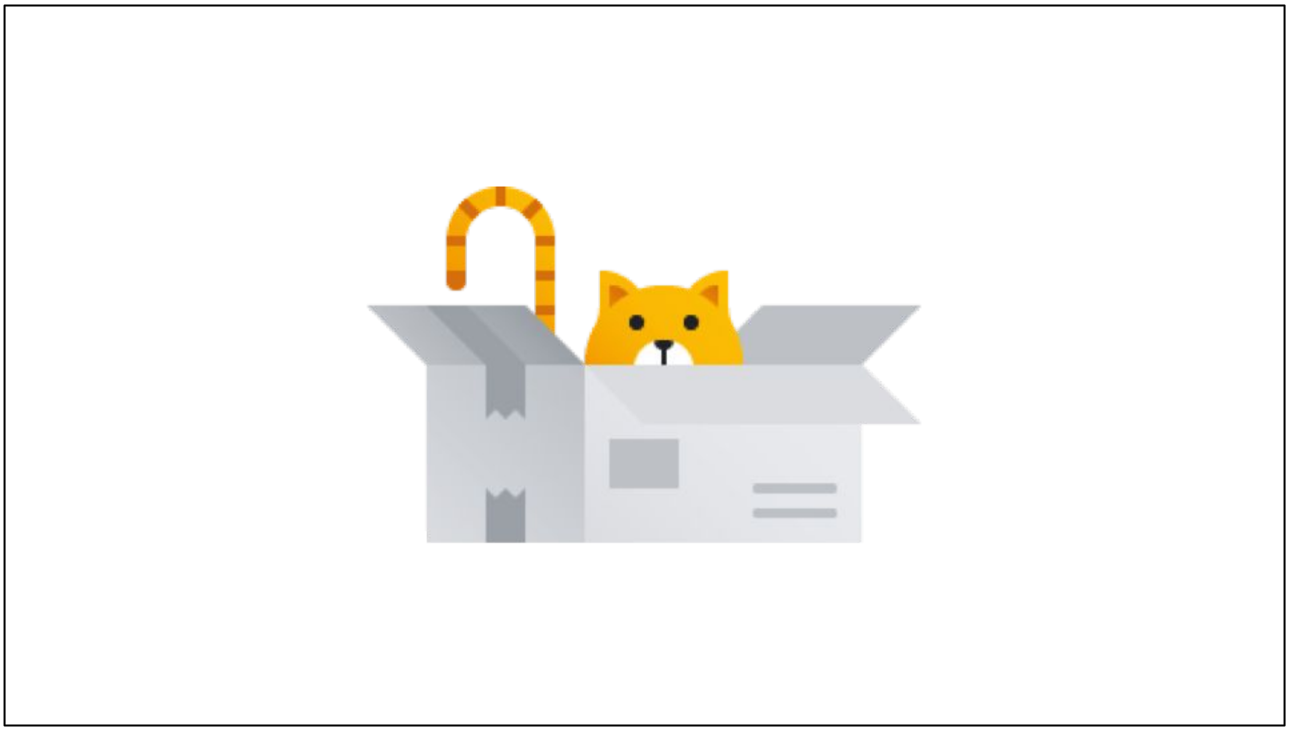
80%
Time spent in top 3 apps

comSore 2015 Mobile App Report

- Building on the trends of multiple years of data, the comScore 2015 Mobile App Report, painted a stark picture of web usage on phones: users spent just 13% (fragment) of their time on mobile devices in browsers; the rest was in apps. Apps were eating the world.
    - The news wasn't all bad, though. While the majority of *time* was spent in apps, comparing the top 1000 mobile web properties vs the top 1000 apps, the web had (fragment) audiences 2.5x larger and grew twice as fast.
    - But while it was harder to build an app audience, app users spent (fragment) 18x more time in their apps than mobile web visitors did.
    - Apps also seemed to be a zero-sum game; (fragment) 4 out of 5 smartphone app minutes were spent on an individual's top 3 apps. Tablets, even more so.
- The big takeaways comSore had from this data were that:
    - It's easier to build a larger audience on the web quicker because of the web's superpower: linking
    - While mobile app usage was exploding, that wasn't at the *expense* of the mobile web or desktops, so they believed the key growth opportunity was in multi-platform engagement
    - Finally, app usage is a reflexive, habitual behavior where those occupying the best home screen real estate were used the most frequently.
- The web at this point had already more or less won Desktop for most usecases: web based email versus a desktop app, anyone? And it was

- already easier to build a larger audience faster on the web than apps; linking and a single shared platform across mobile and desktop meant distribution was much easier. What was the web missing to stay competitive with mobile apps? It lacked an app model.
- The 2015 comScore Mobile App Report came about at just the right time, as the pieces needed to let a site be added to a user's home screen were finally falling into place.

- The missing piece to the app model puzzle came December 17, 2013 with the publication of the first working draft of Manifest for web apps and bookmarks
- The spec, now known as the Web Application Manifest, was initially spec'd by Marcos Caceres from Mozilla and Anssi Kostiainen and Kenneth R Christiansen from Intel.
    - It was designed to consolidate metadata about a web app, like name, icons, and URL to open when launched, from various meta tags into a single place
    - It’s goal: to "enhance bookmarking capabilities such as being able to add a web application to the homescreen of a device"



- The work stemmed from a now-obsolete set of spec work around allowing web apps to be packaged into widgets.
- Those specs, first authored November 9, 2006, defined a widget much like how we'd think of a PWA today:
    - A small client-side application packaged to allow for a single download and installation on a device that may exclude typical web browser interfaces.
    - The Widget family of specs included key aspects needed to add a site to a device, packaging and configuration.
- It… didn't succeed. Even now, you'd be hard-pressed to find mentions of people using it, or it in general outside of talking about the spec specifically. The Cordova (previously PhoneGap) docs mention it, and there's a blog post about using them to write a hybrid app for Tizen, but that's about it. It's legacy, though, would live on.

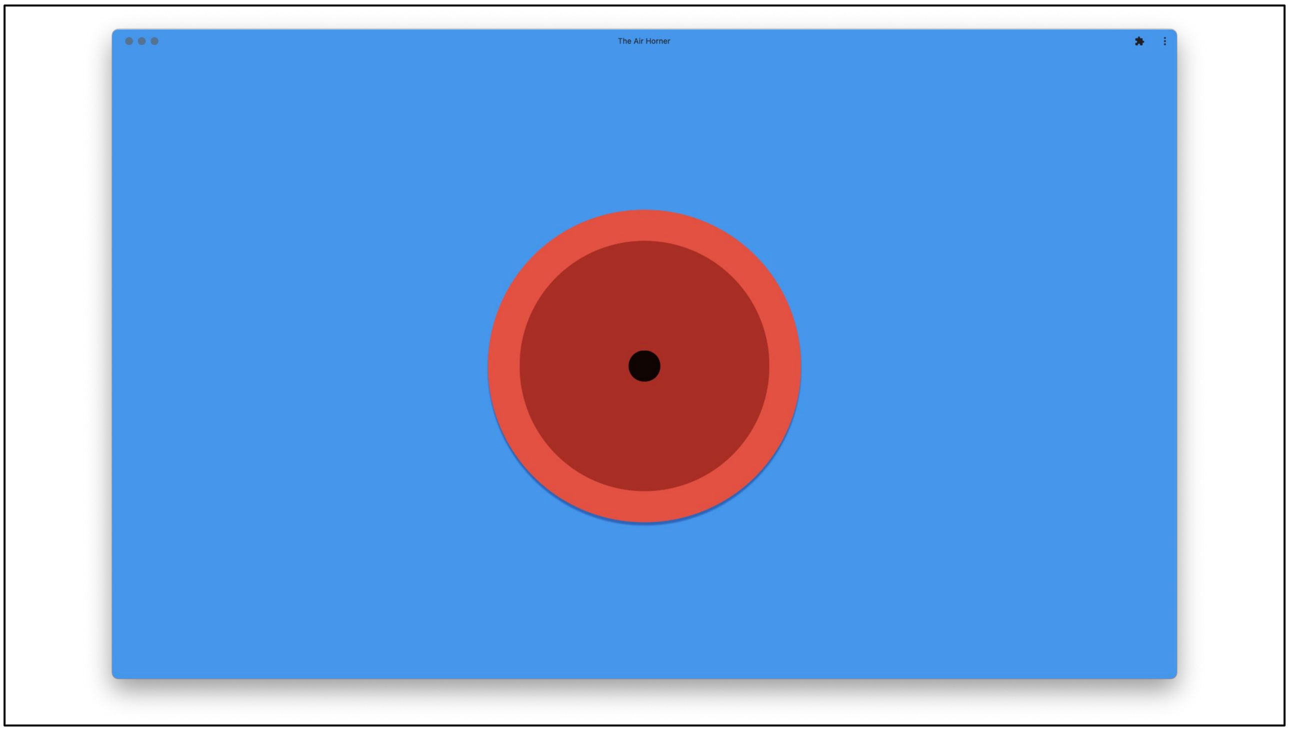
- Kenneth convinced Anssi and Marcos, who was the editor of the Widgets spec, to work together on a runtime spec, later dropped in favor of Service Workers. The bit that was missing, though, was a way to describe the app, and so they pivoted to work on what is now the Web App Manifest spec.
- On mobile devices, Chrome would add support to add web apps to the home screen, with a web app manifest and a service worker, in Chrome 39, November of, 2014
  - Samsung 4 would follow April 2016, and it would arrive for Safari 11.3 March 2018.
  - Less than a year later, on October 11, 2018, the Widget spec would officially be marked as obsolete in favor of service workers with a web app manifest.
- With the launch of Web App Manifests, including key features like start_url and icons, the web now had installation criteria for adding a website to the home screen. That criteria hasn't changed much since.
- But it didn't have a name. They were just "installable web apps".

PWA

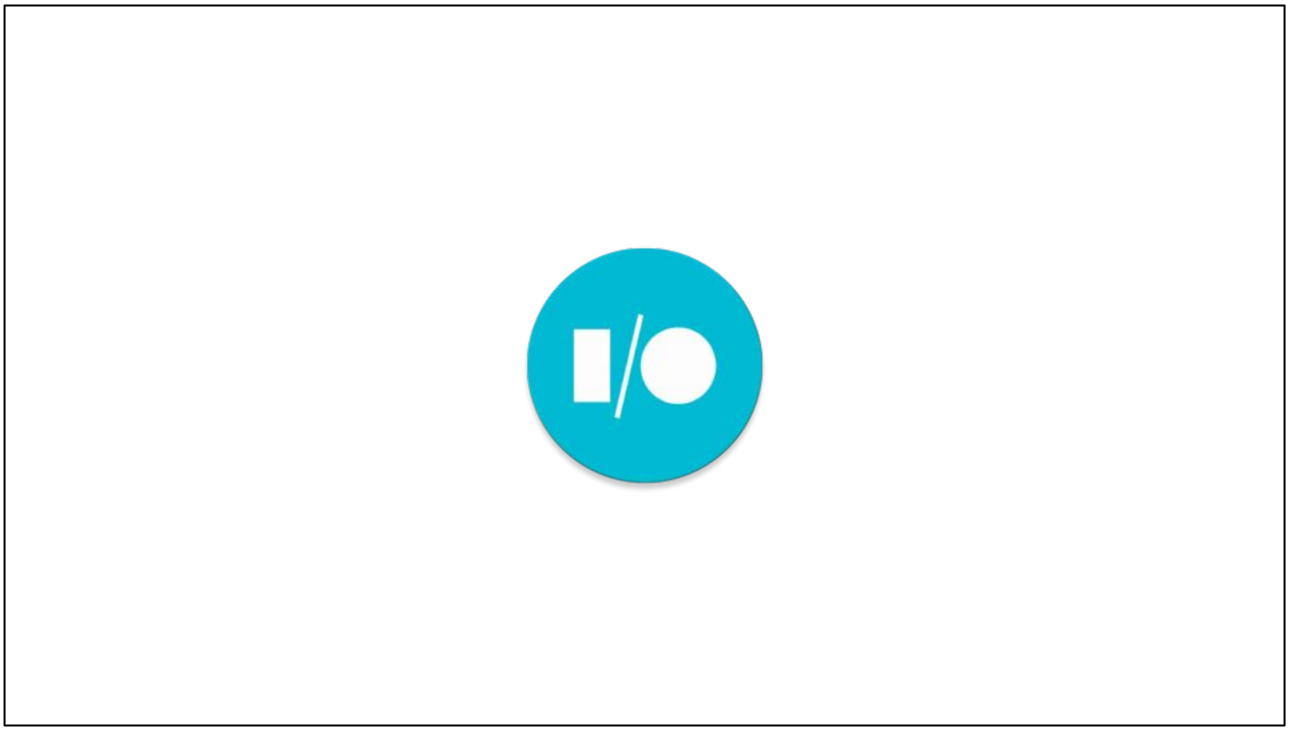
- On June 15, 2015, Alex Russell and his partner Frances would coin the term Progressive Web App to describe web apps that were as reliable as, and could be installed like, mobile apps. A normal website that, when built well and earns a place on your homescreen, could progressively become an "app".
- The name stuck

The First PWA

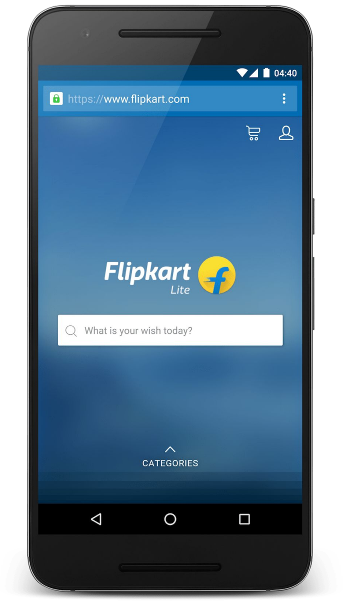
- Half-way through 2015, all the pieces were in place for an app model for the web
    - It had service workers to allow sites to work offline or with limited connectivity
    - It had the web app manifest to describe how an installed site should function
    - And it had a name: Progressive Web Apps.
- It was only a matter of time before one existed.



- The first PWA I can recall, and maybe the first one you can too, is [The Air Horner]() released by Paul Kinlan midway through 2015.
- While fun, it's mostly just a demo.



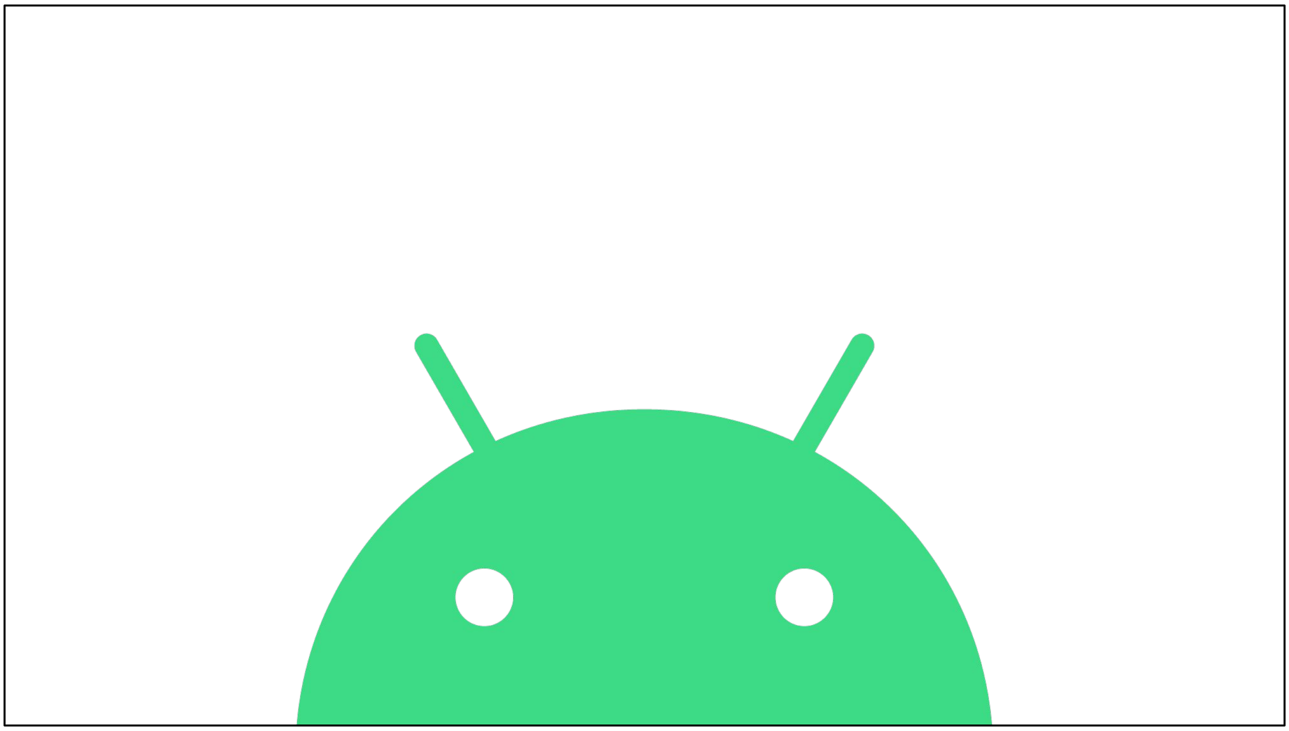
- Or, it may be the Google I/O 2015 website, which was probably the first large-scale PWA and whose development motivated the creation of sw-precache and sw-toolkit, what would go on to become Workbox.



- But, the world's first production PWA is generally considered to be Flipkart Lite by Indian e-commerce company Flipkart, released November 9, 2015.
- Introducing their PWA had a dramatic effect on their user engagement
    - A key metric for Flipkart at the time was tracking data usage to complete first transaction.
        - When comparing Flipkart Lite to the mobile app, Flipkart Lite used **3x less data**.
        - This was especially important as **63% of Flipkart Lite users were on a 2G network**.
        - Service workers and the low friction of the web's distribution model combined to make these possible.
    - Getting Flipkart Lite onto the homescreen was important, too.
        - **60% of all visits** came from launching from the homescreen, and those users **converted 70% more than average users**.
        - Those two activities alone **grew overall engagement 40% higher** than it was before.
- Flipkart proved that PWAs could compete with mobile apps when installed. But e-commerce is a space the web was already good at. To be a truly competitive platform on mobile, the web needed to be able to take on platform apps on their home turf.

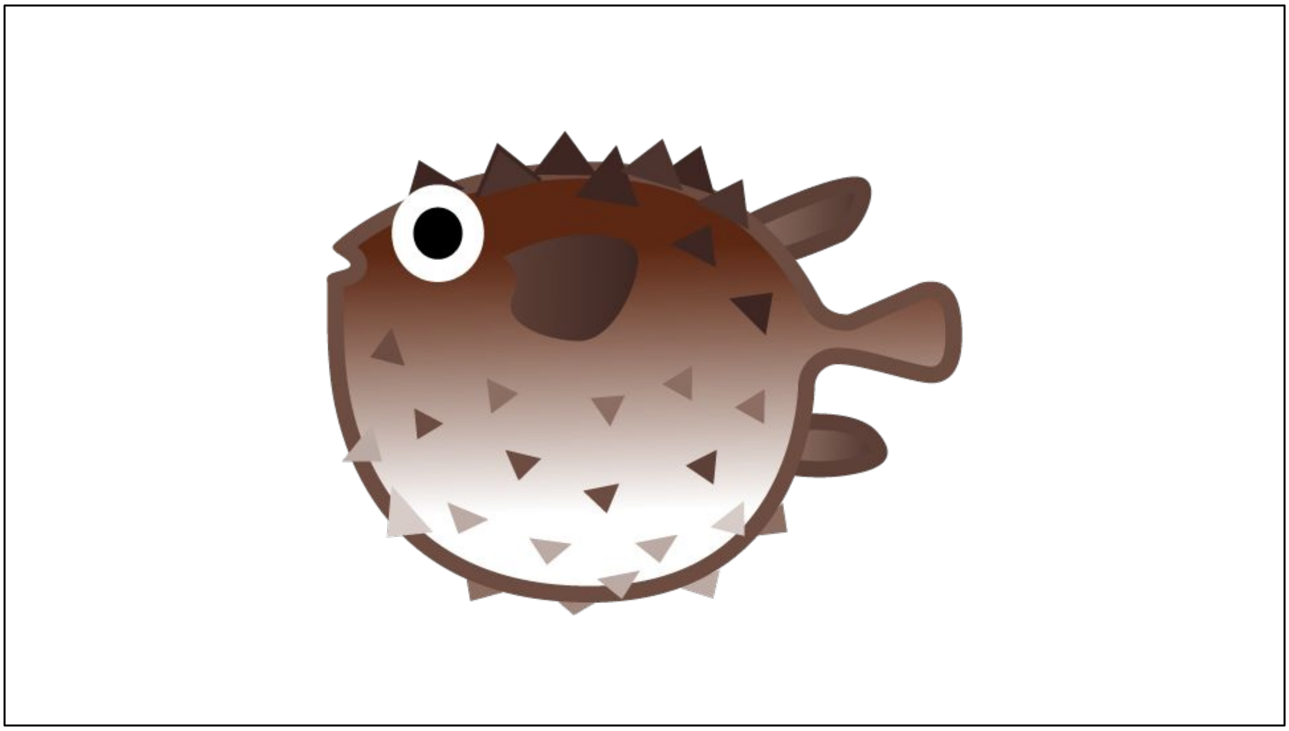
Making PWAs Capable

- "When the web is capable of delivering a particular experience, it can compete." - Alex Russell Progressive Web App Summit 2016
- The web's distribution model is better than app stores, but by design it was less capable than desktop or mobile apps. The drive-by nature of the web made exposing many APIs and features common to platform apps a security risk if they were exposed in the same way. A change in how PWAs were installed paved the way for a new set of work to make the web more capable



- The initial implementation of PWAs on Android was by design minimal; basically they were glorified home screen shortcuts. They didn't show up in the app drawer, they didn't show up in the app switcher, they didn't really "feel" like apps besides having a homescreen icon.
- On Android, the base unit of administration for an app was an APK, or an Android Application Package. The APK is how Android dealt with things like:
  - Notifications
  - Running in the background
  - Targeting for sharing
  - Backup and restoration
  - And more!
- After the successful launch of Flipkart Lite and momentum behind PWAs started to build, the lack of integration of the initial implementation needed to be fixed. On May 10, 2016, the first CL for Web APKs was merged into Chromium.
- WebAPKs were a big deal for how Progressive Web Apps would evolve. Dominick Ng, a lead for Progressive Web Apps at Google, shared: "WebAPKs really built the foundation of the deep system integration that we've pursued ever since." That deep system integration started life as as Project Fizz.

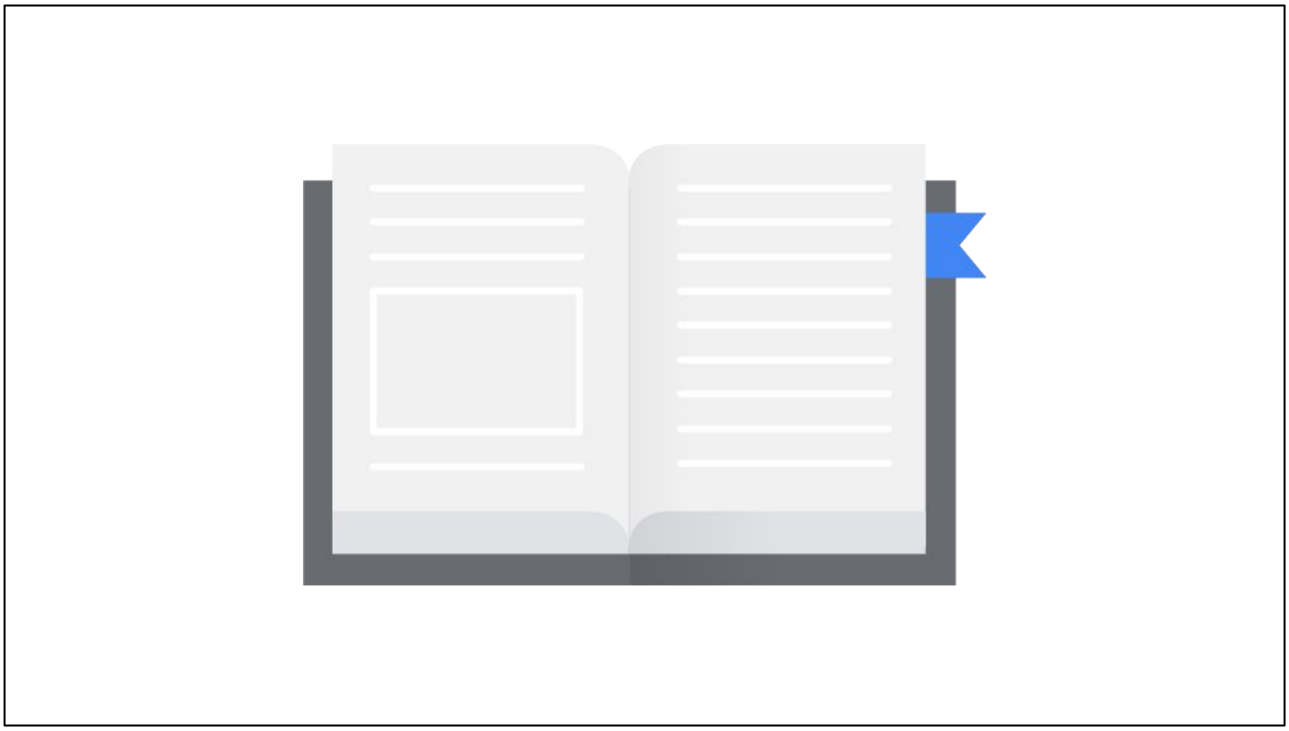
- Project Fizz, named to compliment GIN, an early codename for Chrome, had been around since 2014, not as a formal team, but as a collection of teams within Google, with the goals of:
    - Bringing top-requested APIs to the open web
    - Creating infrastructure that enables new capabilities to be experimented with andexposed in risk-limiting ways.
    - Optionally enable web apps to receive the same UI treatment and integration as mobile and desktop apps
- Fizz was responsible for a lot of early PWA work, including Service Workers, Background Sync and Push Notifications (enabled with the availability of WebAPKs).
- The first two powerful APIs that Fizz introduced, though, were Web Bluetooth, first available in Chrome 56 released January 2017, and Web USB, first available in Chrome 61, released September 2017.
- Around the same time, in late 2015, the Chrome Remote Desktop team, remember them, started planning for the upcoming deprecation of Chrome Apps.
    - The app required a lot of Chrome-specific APIs that they needed to migrate. A large part of their strategy? Write specs for them!
    - That team's work includes specs for the **.code** and **.key** properties of the KeyboardEvent and the async clipboard API.



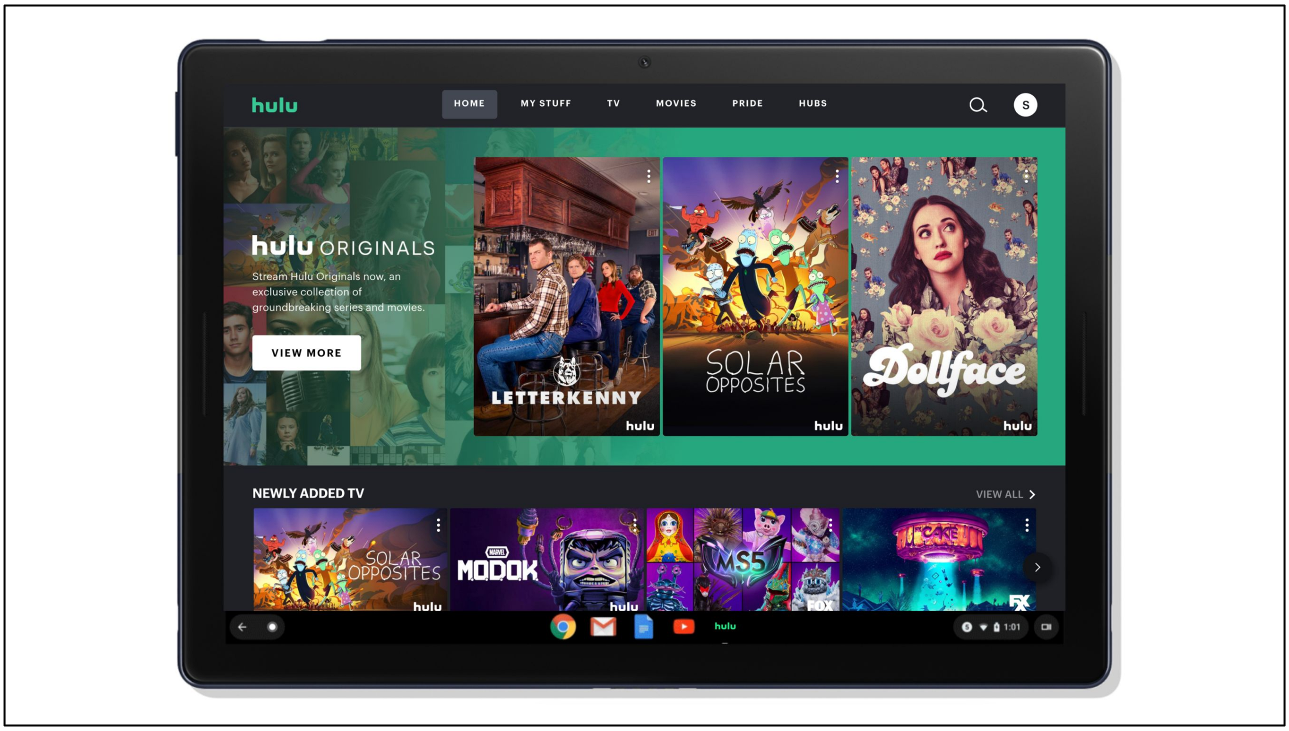
- Fizz would evolve into Project Fugu and the current Capabilities project, pretty much keeping the same mission.
- It would also move beyond Google to a cross-company effort, including significant contributions from Microsoft and Intel.

From Tab to Taskbar

- When Progressive Web Apps were first launched, they were originally envisioned only for mobile devices as a way of helping the web compete with mobile apps.
    - But the web is more than mobile! The benefits of PWAs extend not just to homescreens, but to taskbars too!
    - By allowing PWAs to be installed on desktop devices, the promise of write once, install everywhere could start to be fulfilled.



- Prior to PWAs coming to desktops, Chrome had something called Bookmark Apps
    - Built on the foundation laid by hosted Chrome Apps, bookmark apps, so called because they were something between a bookmark and an app, were what was created if you added a site to your desktop.
    - In 2018, when Chrome started work on bringing true Progressive Web Apps to desktops, the implementation was forked from these Bookmark Apps.
- Rollout of desktop PWAs started with Chrome OS, version 67, released May 2018
    - It would take another 3 releases, to Chrome 70 released October 2018, to add Windows and Linux support
    - And yet another 3 releases, to Chrome 73 on March 2019, for Mac to get support
- Much like on mobile, desktop PWAs, from the start, proved that websites could earn their place on taskbars, often without much effort.

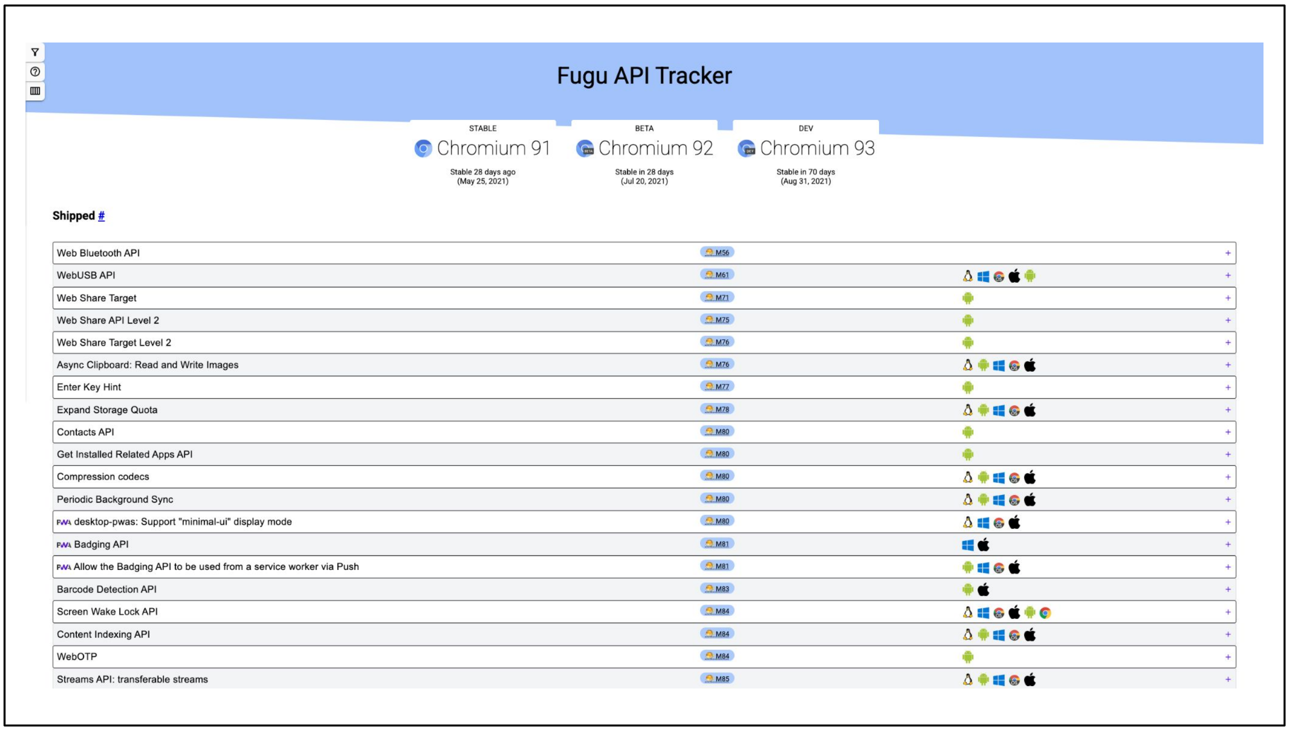


- At Google I/O 2019, Hulu shared their story of creating a desktop PWA to replace a legacy Windows desktop app they had
  - Just like comScore said about mobile back in 2015, Hulu reiterated that, even though the legacy app had poor reviews and lacked features, users kept using it because it was ever-present, just a single tap or click away
  - They started small, with a simple offline fallback service worker and a manifest file; just enough to make their replacement installable.
  - It took 1 developer 2 weeks to build, QA, and release their new experience. That was one sprint for their team, literally the fastest they could possibly deliver results.
  - In the 3 months since launch, they saw 96% of their legacy app users adopt the new PWA, and compared to their website, saw a 27% increase in return visits and a 5.5% increase in engagement.
- Desktop PWAs would come to Edge 79, January 2020.

Today

- And that brings us to today.

- PWAs are here, can be installed on any device with a web browser, and there's lots of work going into making them even more capable, letting the web compete more and more with desktop and mobile apps.



Fugu API Tracker

STABLE	BETA	DEV
Chromium 91	Chromium 92	Chromium 93
Stable 28 days ago (May 25, 2021)	Stable in 28 days (Jul 20, 2021)	Stable in 70 days (Aug 31, 2021)

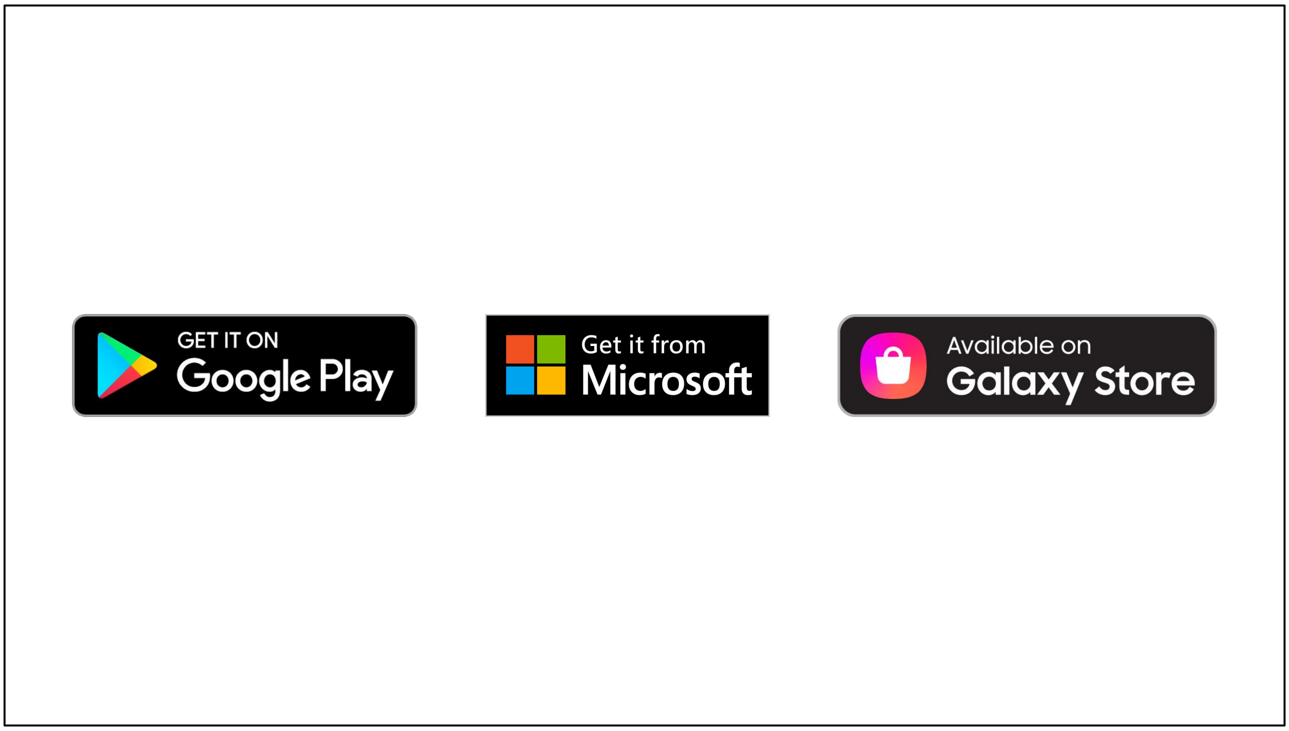
Shipped #

Web Bluetooth API	M56		+
WebUSB API	M61	🐧 🪟 🌐 🍎 🤖	+
Web Share Target	M71	🤖	+
Web Share API Level 2	M75	🤖	+
Web Share Target Level 2	M76	🤖	+
Async Clipboard: Read and Write Images	M76	🐧 🤖 🪟 🌐 🍎	+
Enter Key Hint	M77	🤖	+
Expand Storage Quota	M78	🐧 🤖 🪟 🌐 🍎	+
Contacts API	M80	🤖	+
Get Installed Related Apps API	M80	🤖	+
Compression codecs	M80	🐧 🤖 🪟 🌐 🍎	+
Periodic Background Sync	M80	🐧 🤖 🪟 🌐 🍎	+
PWA desktop-pwas: Support "minimal-ui" display mode	M80	🐧 🪟 🌐 🍎	+
PWA Badging API	M81	🪟 🍎	+
PWA Allow the Badging API to be used from a service worker via Push	M81	🤖 🪟 🌐 🍎	+
Barcode Detection API	M83	🤖 🍎	+
Screen Wake Lock API	M84	🐧 🪟 🌐 🍎 🤖 🌐	+
Content Indexing API	M84	🐧 🤖 🪟 🌐 🍎	+
WebOTP	M84	🤖	+
Streams API: transferable streams	M85	🐧 🤖 🪟 🌐 🍎	+

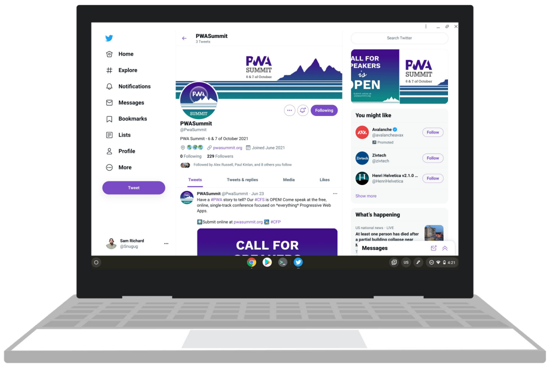
- The Capabilities project has launched 30 new APIs between Chromium 71 and 91, including:
    - Expanded hardware support with WebHID, Web Serial, and Web NFC
    - Deeper OS integration with File System Access, a Contact Picker API, and Web Share and Web Share Target
    - And has even expanded what installed PWAs are capable of, with the badging API and app shortcuts.

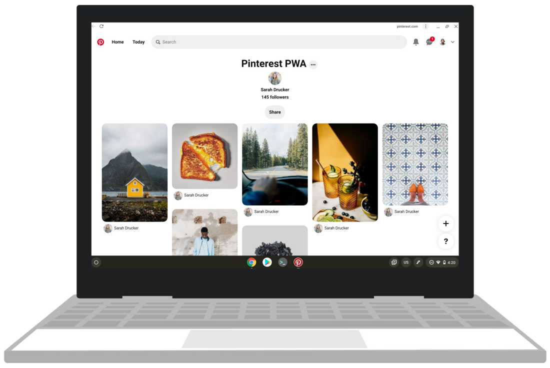
- Modern meta-frameworks, including Next for React, Nuxt for Vue, and Svelte Kit for Svelte, all have built-in or officially supported extensions for creating PWAs, including zero-config options.

- There are developer tools, like Lighthouse, that can be used to test if your website meets installation criteria (and more!) including as part of your CI/CD process
- There are libraries like Workbox to help you build and maintain your service workers, allowing you to quickly implement and leverage common caching and serving patterns
- And there are tools like Bubblewrap, and those powered by it like PWA Builder, to simplify the process of bundling your app for distribution in stores.

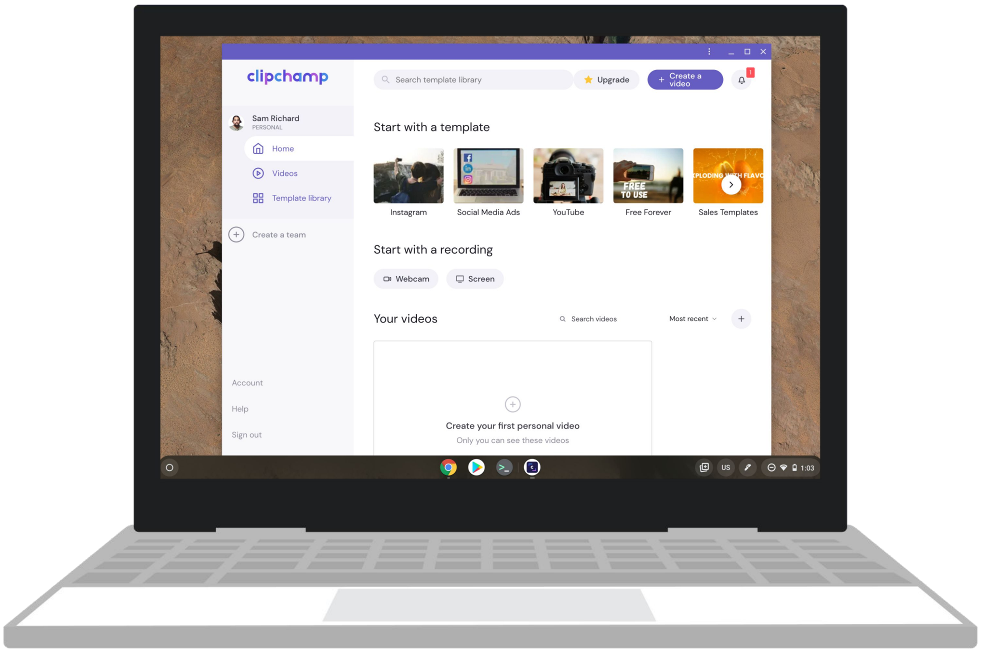


- Speaking of stores, Google Play, the Microsoft Store, and Samsung's Galaxy store all support listing PWAs, letting them take part in the app store discovery, preview, and review ecosystem

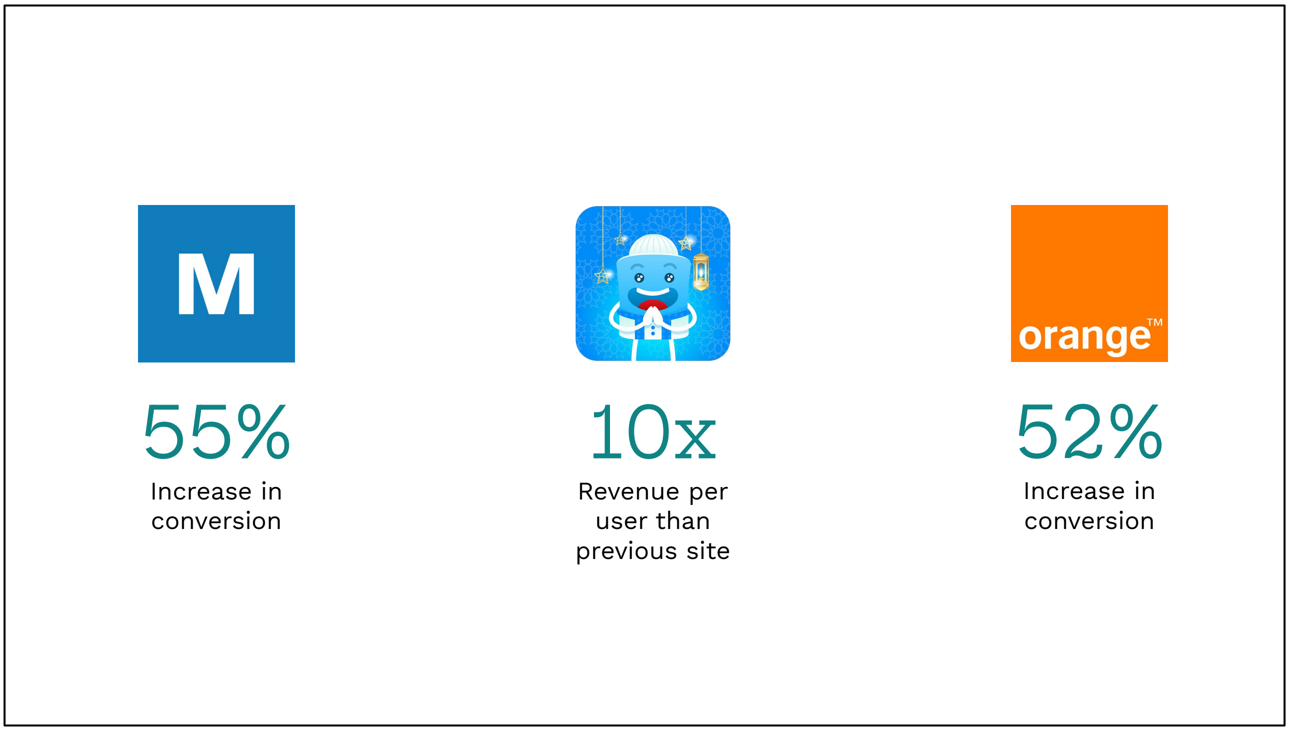




- On Chrome OS, for instance, both Twitter’s and Pinterest’s PWA experiences are so good that they’ve both replaced their Android apps in the Chrome OS Play store with their PWAs



- Clipchamp, an in-browser video editor, had never been available in an app store before.
- After upgrading to a PWA and getting listed in the Chrome OS Play store, they've seen PWA installation rates increasing by 97% a month in their first five months, with PWA users having a 9% higher retention rate than standard desktop web users.
- Their video editor? It also got an overhaul, moving from NaCl to Web Assembly,



55%

Increase in conversion

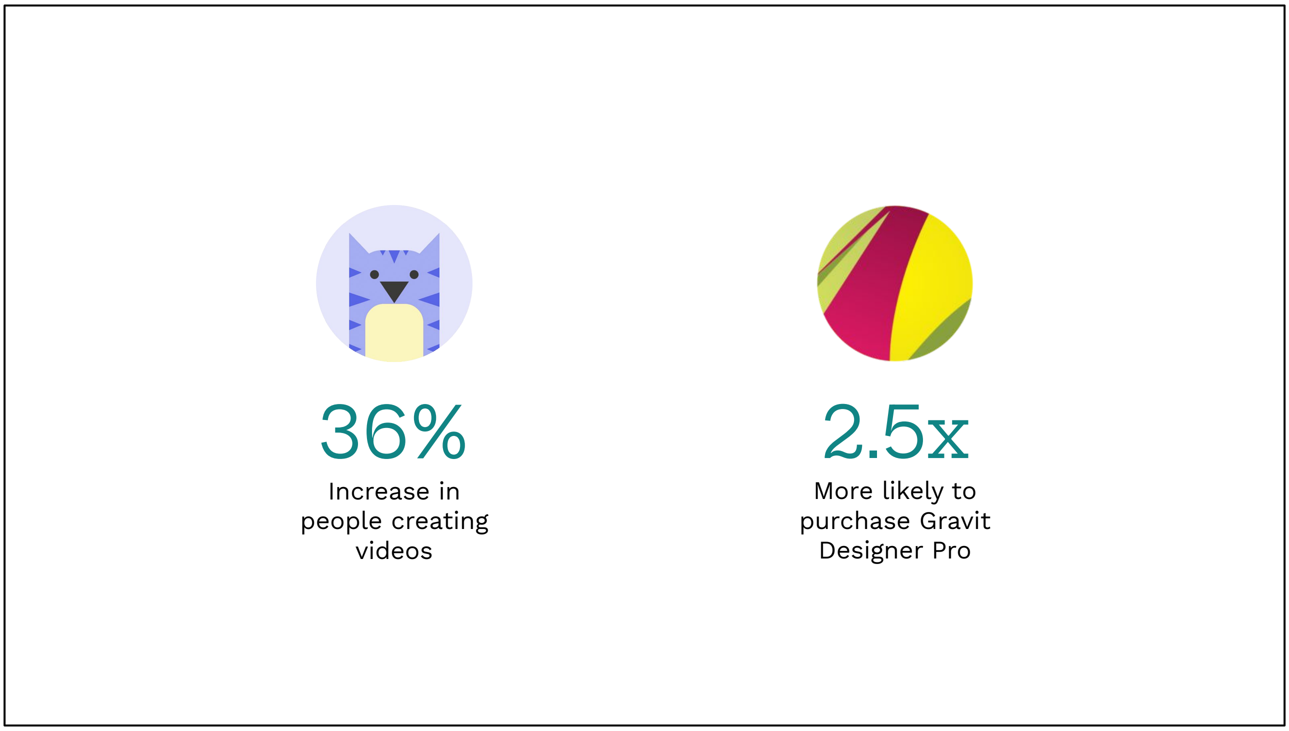
10x

Revenue per user than previous site

52%

Increase in conversion

- Clipchamp's success with PWAs isn't unique. On mobile…
- From a May 2021 case study, Mainline Menswear, a UK-based online clothing retailer, saw a 55% increase in conversion rate for their PWA versus their mobile site, along with a 12.5% higher average order value order value and 243% higher revenue per session.
- From a June 2021 case study, Blibli, an e-commerce marketplace in Indonesia, found their PWA genreates 10x more revenue per user than their previous mobile site, reduces bounce rate by 42%, and shows an 8x improvement in mobile conversation rate.
- And from a May 2021 case study, Orange Polska S.A., a leading telecommunications service in Poland, saw a 52% increase in conversion rate for their PWA versus their previous mobile site, along with 30% faster average loading time and an 18% increase in session depth.



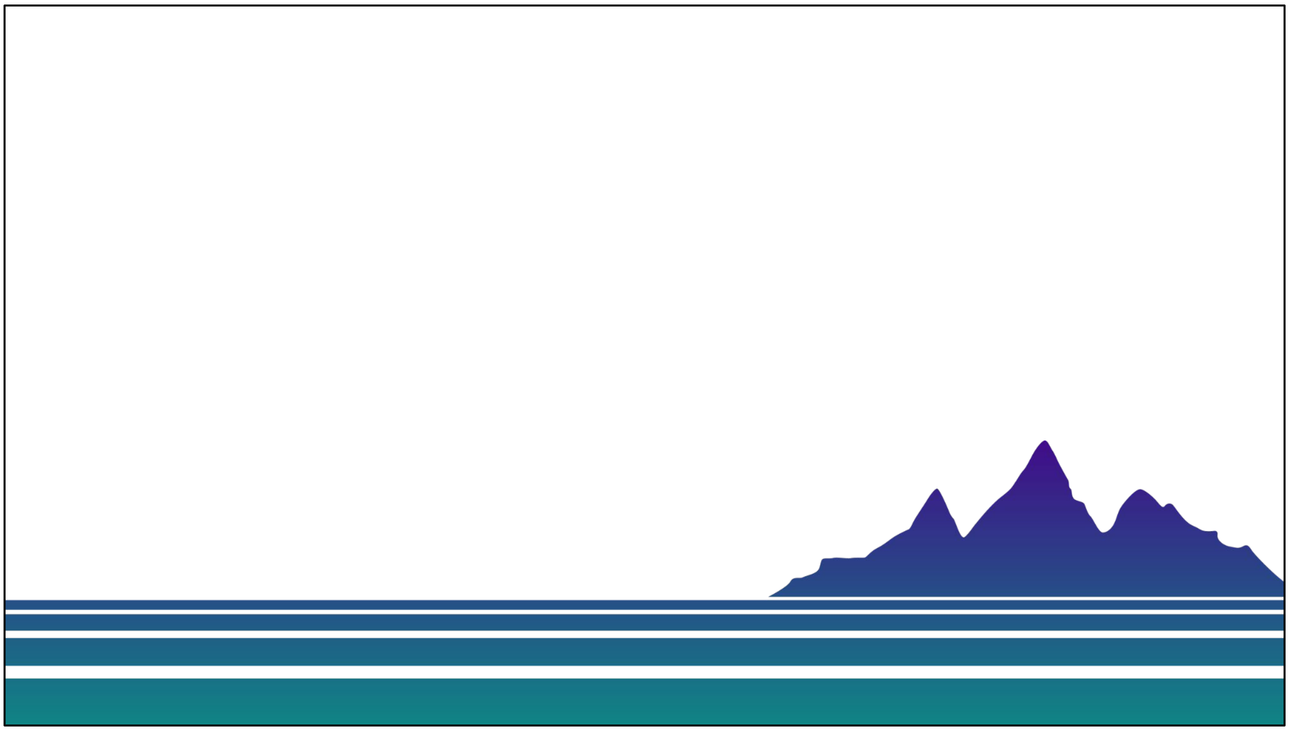
36%

Increase in
people creating
videos

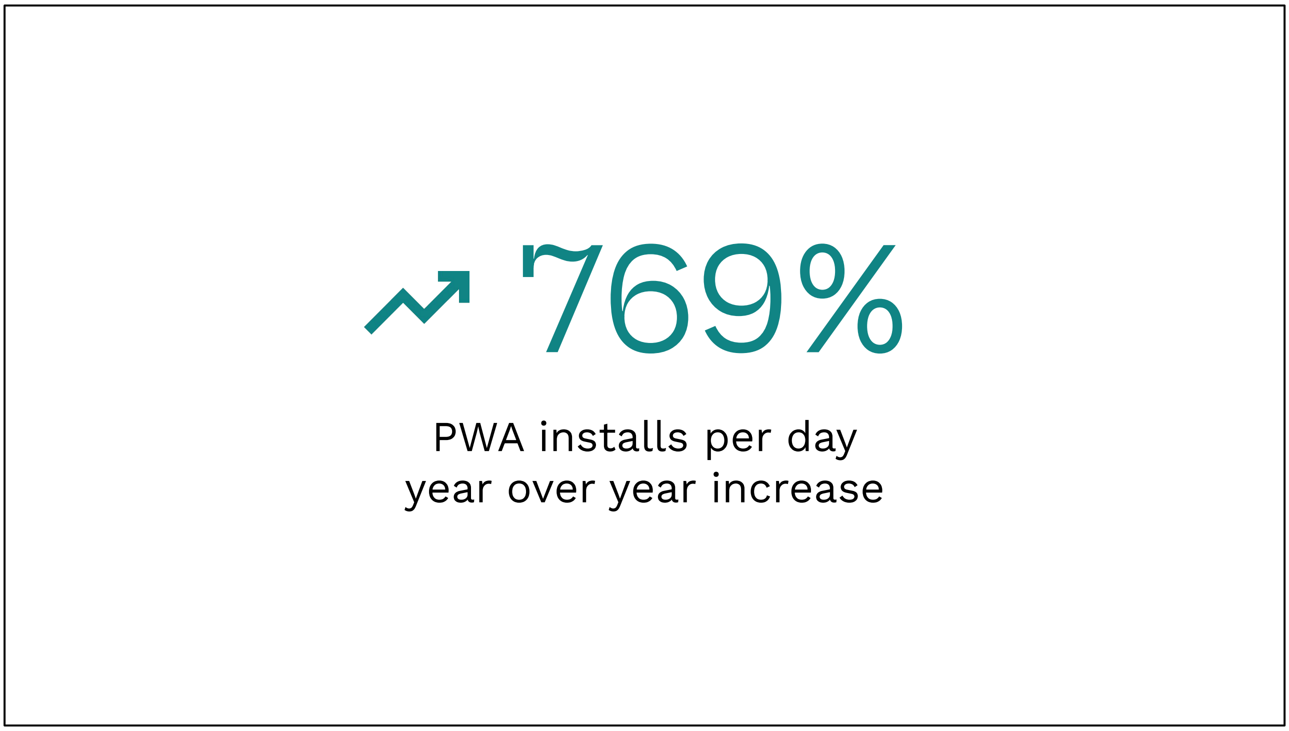
2.5x

More likely to
purchase Gravit
Designer Pro

- On desktop, we’re seeing the same thing…
- From a December 2020 blog post, Kapwing, an online image, video, and Gif editing platform, saw that within the first five weeks of launching their PWA-ified website, the number of people creating videos through the PWA grew 36%, outpacing overall website growth and therefore indicating a higher retention rate among creators who installed the PWA.
- And from a December 2020 case study, Corel Corporation’s Gravit Designer Pro, a powerful vector design tool that also offers Windows, Mac, and Linux apps, saw PWA users are 24% more active than any other install type, account for 31% more repeat users than other platforms, and are 2.5x more likely to purchase Gravit Designer Pro.



- Getting to today has been like climbing a mountain: it's taken a while, there have been lots of up and downs, but now that we're at the summit, we can reflect on what we've learned
    - When they were first introduced, PWAs were about push notifications, working offline, and being installable on phones.
    - There was a push to make your whole site work offline
    - And an App Shell model that favored single-page apps, and refactors, was encouraged.
    - Today, we know that any way you want to build your site, single-page or multi-page app, as a static site or a dynamic one, are all great options for building your PWA
    - We've seen that with even minimal work, adding a custom offline page and a web app manifest to make a site installable, produces big results without needing to re-architect your site, add lots of features, or ensure everything works offline..
    - And we've learned that, outside chat and social networks, basically no one likes push notifications.



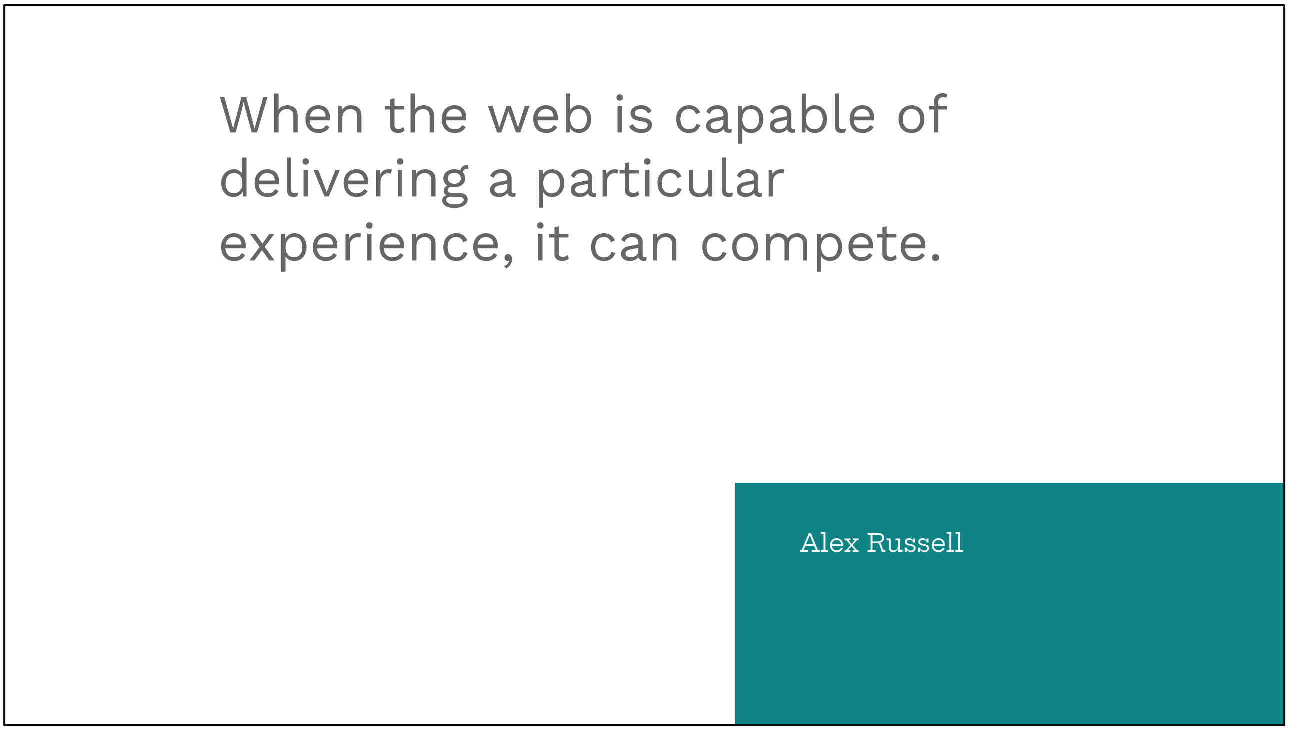
↗ 769%

PWA installs per day
year over year increase

- We've seen total PWA installs per day increase by 769% YoY
- Today, it's no longer a question of if you can, or should you, make your site a PWA. It's a question of why haven't you yet.

The Future

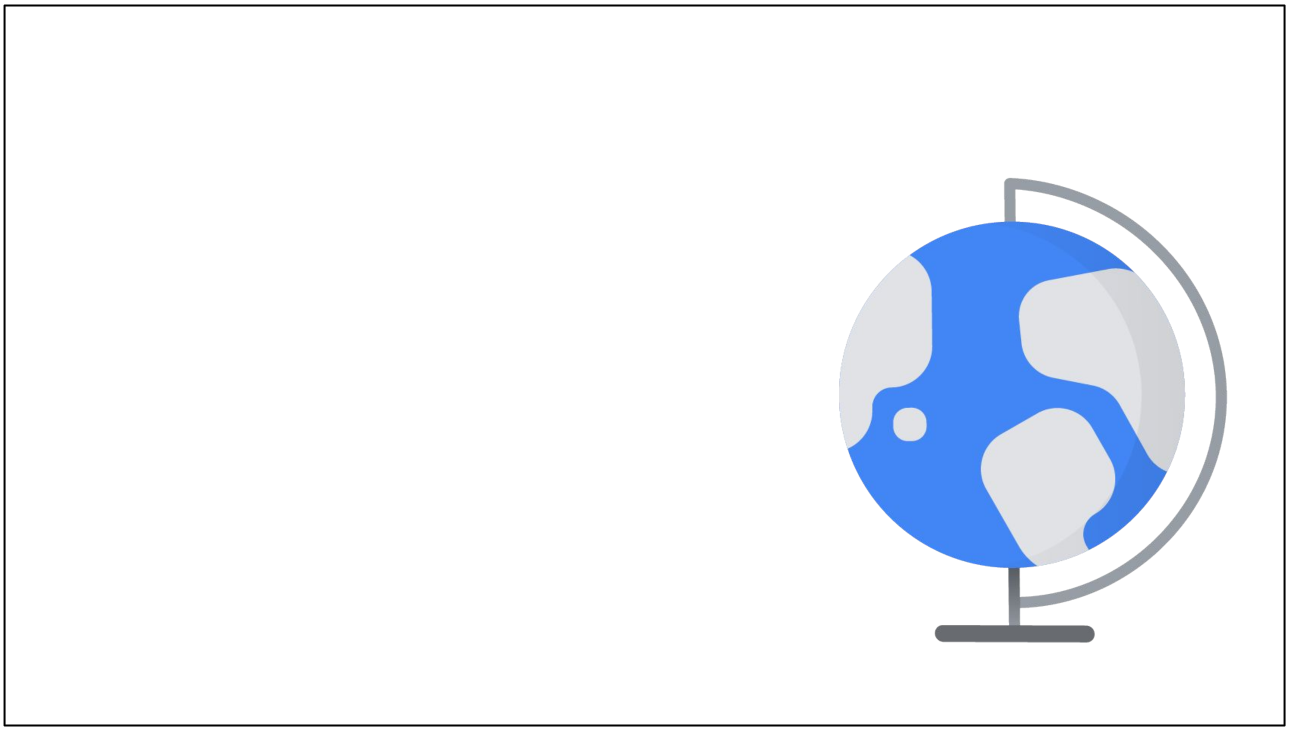
- Maybe that's not fair. Sure, making a site installable is now easier than ever, and there are benefits from even that little bit of work, but is that enough?



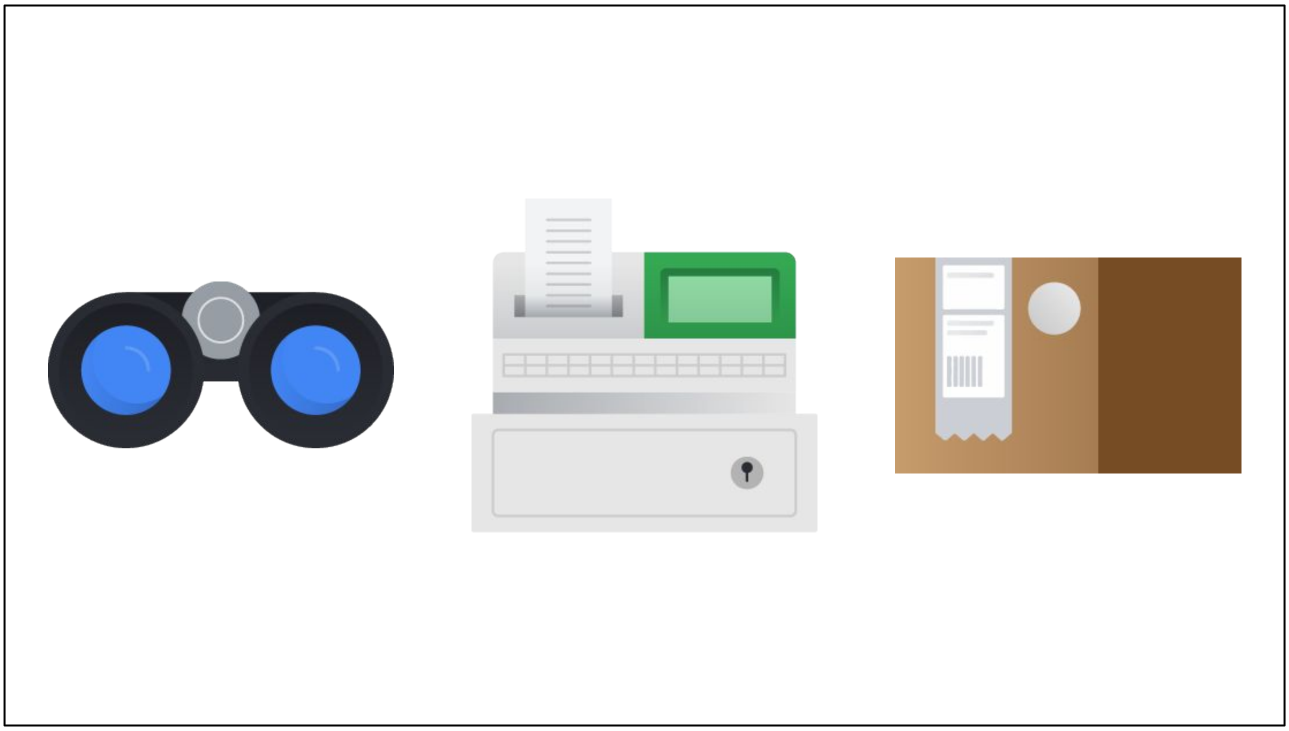
When the web is capable of delivering a particular experience, it can compete.

Alex Russell

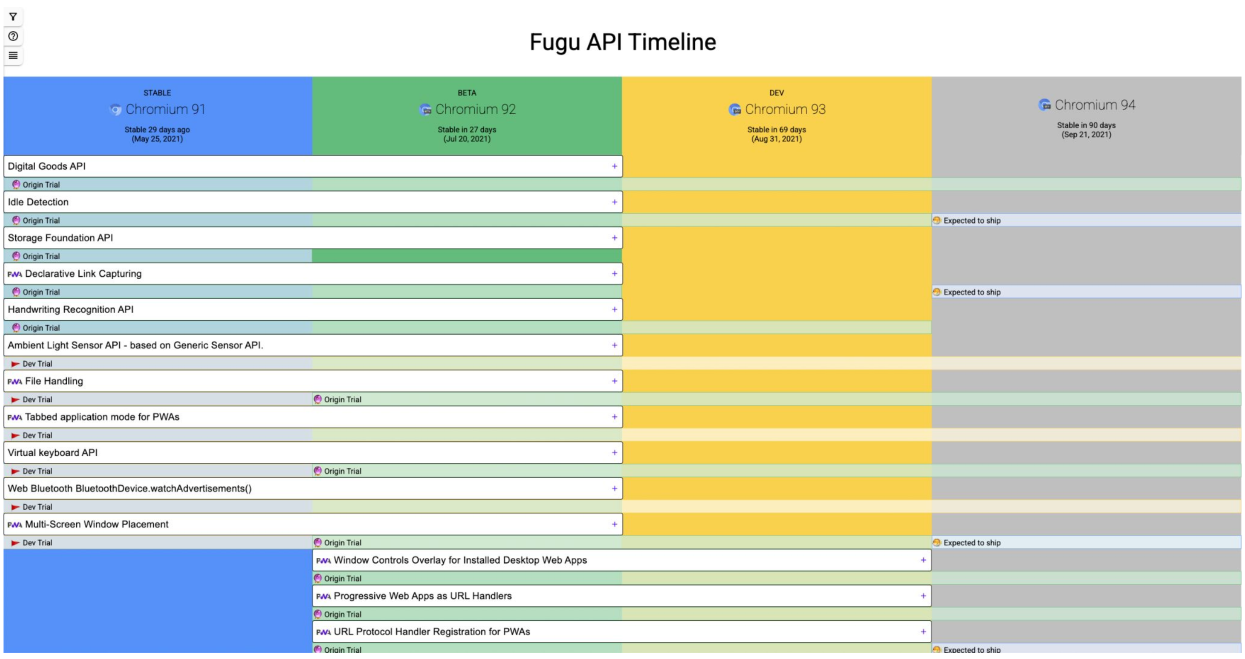
- (read quote)
- So I ask you, take a moment to think about it. What’s holding your work back from being able to compete? What’s the web you want? (10 second pause)
- For me, I think this is the future of Progressive Web Apps. Taking that thought of the web as an application platform from the first draft of HTML5 all the way back in 2005 and making it a reality.



- PWAs can, and should, mean more than an installable website that doesn't crash when offline. PWAs should be the lens through which we view the web as the only truly universal application platform.
- Create once, deploy everywhere. One codebase, one team, one skillset, building one app that's installable, reliable, and capable, regardless of what device a user chooses to access it from.
- To get there, we need to get better at delivering more experiences. And there are some experiences where the web doesn't compete well today.



- App discovery on the web isn't great.
    - For better or worse, users expect to find, preview, and review apps from an app store.
    - Some users find curated stores, like Google Play or Apple's App Store, a more trustworthy way to find apps than just a web search.
- App stores also have built-in monetization tools that the web lacks.
    - Upcoming APIs, like the Digital Goods API, coupled with the Payment Request API, are looking to make a dent on this for the web, but it's still a tough, not-yet solved problem.
- Packaging is still a problem;
    - How can we install a PWA without an initial network connection?
    - Is it possible to provide an installer for PWAs like desktop apps traditionally have?
    - How can we trust the contents of those packages?



Fugu API Timeline

STABLE Chromium 91 Stable 29 days ago (May 25, 2021)	BETA Chromium 92 Stable in 27 days (Jul 20, 2021)	DEV Chromium 93 Stable in 69 days (Aug 31, 2021)	Chromium 94 Stable in 90 days (Sep 21, 2021)

Digital Goods API
Origin Trial

Idle Detection
Origin Trial — Expected to ship

Storage Foundation API
Origin Trial

Declarative Link Capturing
Origin Trial — Expected to ship

Handwriting Recognition API
Origin Trial

Ambient Light Sensor API - based on Generic Sensor API.
Dev Trial

File Handling
Dev Trial — Origin Trial

Tabbed application mode for PWAs
Dev Trial

Virtual keyboard API
Dev Trial — Origin Trial

Web Bluetooth BluetoothDevice.watchAdvertisements()
Dev Trial

Multi-Screen Window Placement
Dev Trial — Origin Trial — Expected to ship

Window Controls Overlay for Installed Desktop Web Apps
Origin Trial

Progressive Web Apps as URL Handlers
Origin Trial

URL Protocol Handler Registration for PWAs
Origin Trial — Expected to ship

- And then there are capabilities. This, I think more than anything else, is the story of the future of PWAs. The web as an application platform is not going away, it's only growing, and to compete, we need to expand what the web's capable of doing.
    - Work is being done today to let installed PWAs register as file handlers, URL handlers, and register as protocol handlers.
    - There's work being done to enable new data transport options, give more control over PWA app windows, access local fonts, and use the browser's built-in media codecs.
    - Even work being done to enable enterprise usecases, like reading device attributes and establishing custom TCP and UDP connections.
- Developer feedback is what drives these new capabilities. And your feedback is critical. So I ask again:
    - What's holding you back from being able to compete?
    - What does the web need to succeed as an application platform?
- Think about it, then share your thoughts with browser vendors; in their issue queues, in comments on web specs, and through WebWeWant.fyi
- Together, we can make the web a truly unique, and amazing, application platform.

Special Thanks

Special thanks to these individuals for helping fill in the history of PWAs from the unique perspective of those who were there at the time.

Alex Russell, Dominick Ng, Alan Cutter, Yaron Friedman, Alexey Baskakov, Joshua Bell, Gary Kačmarčík, Alexey Baskakov, Glenn Hartmann, Domenic Denicola, Vincent Scheib, and Kenneth R Christiansen

Thank You

- Thank you all very much.