The Capable Web

Thomas Steiner
steiner.thomas@gmail.com
Reutlingen, Germany

ABSTRACT

In this paper, I discuss arguments in favor and in disfavor of building for the Web. I look at three extraordinary examples of apps built for the Web, and analyze reasons their creators provided for doing so. In continuation, I look at the decline of interest in cross-platform app frameworks with the exception of Flutter, which leads me to the two research questions (i) "Why do people not fully bet on PWA" and (ii) "Why is Flutter so popular". My hypothesis for why developers don't more frequently set on the Web is that in many cases they (or their non-technical reporting lines) don't realize how powerful it has become. To counter that, I introduce a Web app and a browser extension that demonstrate the Web's capabilities.

CCS CONCEPTS

• **Information systems → Browsers**.

KEYWORDS

Progressive Web Apps, Project Fugu, browser APIs, Web extensions

ACM Reference Format:
Thomas Steiner. 2023. The Capable Web. In *Proceedings of The Web Conference 2023 (WWW '23).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

If you want to build an app, you have several ways of doing it. You can build a platform-specific app for the platforms you care about, for example, Windows, Android, and iOS. In which case you would build three apps. You can also build a (Progressive) Web App, possibly in addition to platform-specific apps. Alternatively, you can choose a cross-platform framework such as Electron.js][1] or Ionic[2] that promises to let you write once and run anywhere. Let me begin by walking you through three extraordinary examples of apps whose makers chose to also build for the Web, apart from building platform-specific apps.

[1]https://www.electronjs.org/
[2]https://ionicframework.com/

WWW '23, April 30 – May 4, 2023, Austin, TX

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1.1 Beacon cases for the Web's abilities

1.1.1 Photoshop. I always saw Photoshop as one of the last bastions of high quality apps that supposedly would never make it to the Web platform. This last bastion has finally fallen. With Photoshop,[3] Adobe, together with Chromium engineering, has managed to get a beta version of Photoshop running in the browser (see Figure 1) that can serve as the new beacon showcase of what is possible on the Web. In it, you can try out the commenting workflow and test some early Photoshop editing features Adobe is piloting on the Web. You and your collaborators can now open and view Photoshop cloud documents in the browser, provide feedback, and make basic edits. All from the Web without having to download the full Photoshop app for your specific operating system.

1.1.2 Visual Studio Code. Similarly, Microsoft has launched Visual Studio Code[4] on the Web (see Figure 2), a fully fledged, installable Web experience of its integrated development environment (IDE) that makes developing completely in the browser possible, including the option to open and edit files on the local file system.

1.1.3 Twitter. Lastly Twitter—whose Progressive Web App (PWA) is largely seen as probably the best mainstream Progressive Web App[5]—has used its responsive Web codebase for all platforms, mobile and desktop, via Web browsers.[6] On Windows, the PWA is the experience the company is confident enough to make the Twitter experience that you get when you install the app from the Microsoft Store[7] (see Figure 3).

[3]https://web.dev/ps-on-the-web/
[4]https://code.visualstudio.com/blogs/2021/10/20/vscode-dev
[5]https://www.thurrott.com/cloud/social/150171/
[6]https://blog.twitter.com/engineering/en_us/topics/insights/2019/twitter-for-mac-is-coming-back
[7]https://blog.twitter.com/en_us/topics/product/2018/a-new-twitter-experience-on-windows

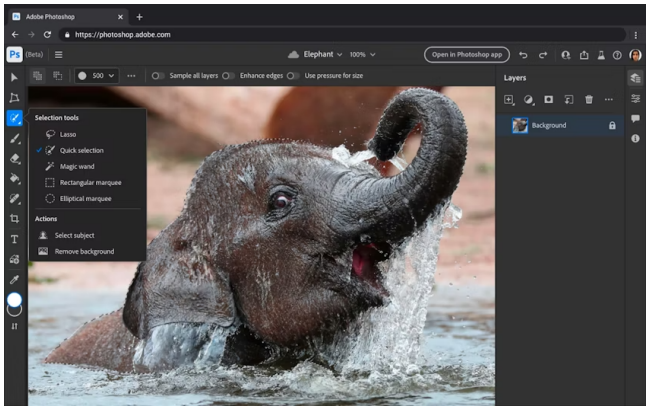

Figure 1: Photoshop on the Web

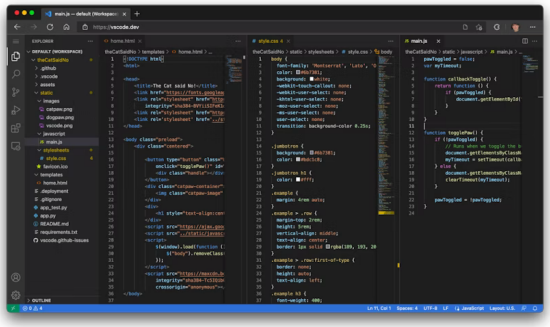

Figure 2: VS Code on the Web

Figure 3: Twitter in the Microsoft Store

2 LINKABILITY AND UNIVERSALITY: THE WEB'S SUPER POWERS

Linkability. All three companies, Adobe, Microsoft, and Twitter, in parallel with their Web apps, have well-established, platform-specific Windows, macOS, Android, iOS, and Linux versions of their apps Photoshop, Visual Studio Code, and Twitter respectively. So why did they build for the Web on top? The answer lies in its linkability and universality.

As Google's Thomas Nattestad put it: *"The simple power of a URL is that anyone can click it and instantly access it. All you need is a browser. There is no need to install an application or worry about what operating system you are running on"*.[8] According to Microsoft's Chris Dias, with Visual Studio Code for the Web, when working with GitHub *"you can make quick edits, review PRs, and continue on to a local clone"*.[9] The sole fact that you can share a link to your work unlocks collaboration patterns that users have embraced and loved since the birth of apps such as Google Docs. Twitter, of course, lives and dies by its links. News sites regularly link to newsworthy tweets, which means *"keeping it quick"*[10] is core to ensuring people can get from an article straight into the app, where they can read or engage with the linked tweet.

Linkability of platform-specific apps. While more ubiquitous on mobile, linking into a platform-specific app from the Web on desktop is comparatively rare. On mobile (and macOS), this works via a technology called Universal Links[11] on iOS (and on macOS), and App Links[12] on Android. Platform-specific apps alternatively can rely on registered protocol schemes[13] such as `itms-apps:` for when you want to deep-link into the App Store app on macOS or iOS, or register your own custom schemes for your own apps. So while technically possible, linking into platform-specific apps is a lot less flexible and requires more plumbing work than simply linking into a Web app.

Universality. Web applications are inherently universal. They run on whatever operating system is capable of running a Web browser and they do not need to be compiled for each operating system separately. The same code base powers the application on all platforms. This doesn't mean that there are no compatibility issues—there are plenty actually—but there is a solid, shared, increasing baseline[14] that all applications can build upon.

3 THE SLOW DECLINE OF INTEREST IN CROSS-PLATFORM APP FRAMEWORKS AND THE RISE OF FLUTTER

The Web isn't the only platform that promises "write once, run anywhere". Cross-platform frameworks à la Electron.js do, too. With the Web becoming powerful enough to drive apps such as Photoshop that were thought to be impossible, we can, however, observe a slow decline of interest in cross-platform desktop app frameworks such as Electron.js and NW.js,[15] and mobile app frameworks such as Cordova[16] or React Native;[17] while at the same time there is an undeniable *increase* of interest in Flutter.[18]

Google Trends stats. The Google Trends chart[19] in Figure 4 shows the five frameworks side by side. While noting that this chart does show disambiguated *topic trends* as detected by Google (as opposed to ambiguous *search term trends*), it's clearly not an exact science.

Statista stats. As shown in Figure 5, this trend is backed by data observed by Statista,[20] according to which Flutter has passed React Native as the most popular framework.

StackOverflow stats. StackOverflow statistics[21] on tag usage also support this, as you can see in Figure 6. The underlying assumption of people actually using a technology correlating with people asking questions about that technology on StackOverflow isn't beyond the realms of possibility.

[8] https://web.dev/ps-on-the-web/
[9] https://code.visualstudio.com/blogs/2021/10/20/vscode-dev
[10] https://blog.twitter.com/engineering/en_us/topics/infrastructure/2019/progressively-enhancing-desktop-devices
[11] https://developer.apple.com/ios/universal-links/
[12] https://developer.android.com/training/app-links/
[13] https://developer.apple.com/documentation/xcode/defining-a-custom-url-scheme-for-your-app
[14] https://web.dev/interop-2022/
[15] https://nwjs.io/
[16] https://cordova.apache.org/
[17] https://reactnative.dev/
[18] https://flutter.dev/
[19] https://goo.gle/google-trends-cross-platform-apps-frameworks
[20] https://www.statista.com/statistics/869224/
[21] https://goo.gle/stackoverflow-cross-platform-apps-frameworks

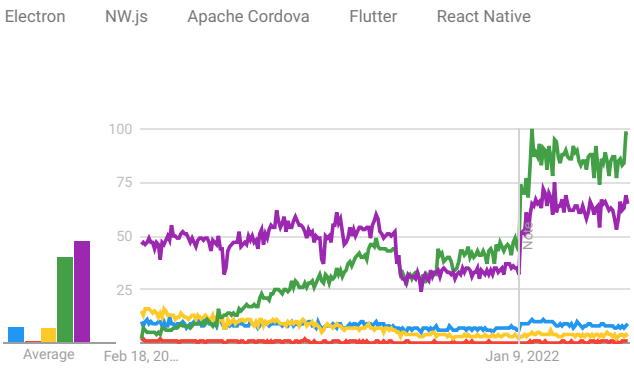

Figure 4: Google Trends interest over time for Electron (blue), NW.js (red), Apache Cordova (yellow), Flutter (green), and React Native (purple)

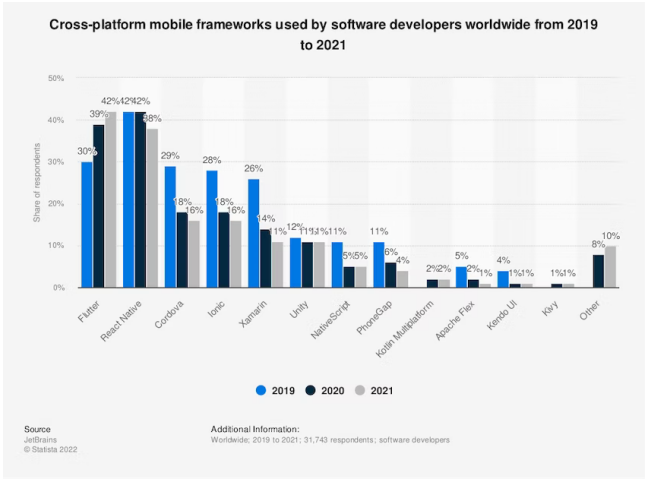

Figure 5: Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021

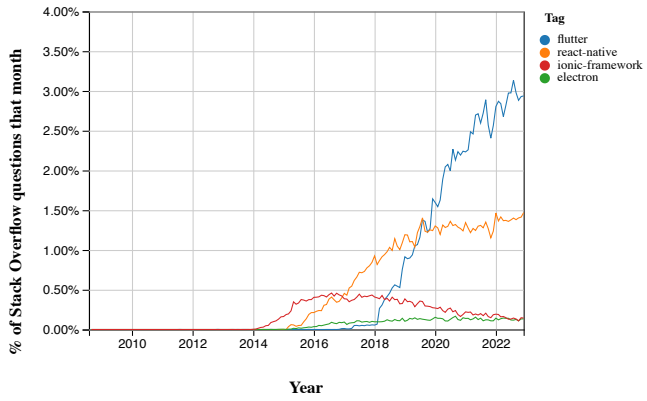

Figure 6: StackOverflow statistics on tag usage

4 RESEARCH QUESTIONS

Given the examples of Photoshop, VS Code, and Twitter, that show that it's indeed possible to build amazing applications on the Web, and given the Web's super powers of linkability and universality plus the decline in cross-platform app frameworks (except Flutter), two research questions come up:

RQ1. Why do people not fully bet on PWA?
RQ2. Why is Flutter so popular?

5 WHY DO PEOPLE NOT FULLY BET ON PWA?

For RQ1, the answer is complex and manifold. I want to break it down into different sub-categories.

5.1 Missing capabilities: aka the app gap

Web applications still lack certain functionalities that platform-specific apps have. In the following paragraph, I list representative examples of such functionalities on different platforms.

First, it's impossible, even with an installed PWA, to obey the macOS user interface paradigm of having the app menu at the top of the screen.[22] It can easily be achieved with frameworks such as Electron.js via the `Menu`[23] class. (On the Web, the next best thing is Window Controls Overlay;[24] getting support for app menus is tracked as https://crbug.com/1295253.) Another example is in-app purchases on macOS that can be handled via Electron's `inAppPurchase()` method.[25] (On the Web, the next best thing is the Digital Goods API,[26] currently limited to Android and ChromeOS.) Installers are a common way users have learned to install applications on Windows. With Electron.js, it's possible to create installers[27] and make installed applications update automatically.[28] (On the Web, Web Bundles[29] are the next best alternative in Chrome.) This list isn't exhaustive, and Electron.js is mentioned as a representative app framework out of several.

How big a challenge is this? There are undeniably a number of capabilities that are missing from the Web. In many cases, they are "nice-to-have", but not necessarily required for a still great experience. Carefully assess if a capability can be seen as a progressive enhancement. For an example of this approach, check out the article Progressively enhance your Progressive Web App.[30]

5.2 Discoverability in app stores

Collectively, we have educated users to look for apps in app stores. Some stores such as the Windows Store[31] and the Android Play Store[32] have started to embrace (wrapped!) PWAs (optionally limited to ChromeOS[33]), and offer graphical user interface tools like

[22] https://developer.apple.com/design/human-interface-guidelines/macos/menus/menu-bar-menus/
[23] https://www.electronjs.org/docs/latest/api/menu
[24] https://web.dev/window-controls-overlay/
[25] https://www.electronjs.org/docs/latest/tutorial/in-app-purchases
[26] https://goo.gle/trusted-web-activity-receive-payments
[27] https://www.electronjs.org/docs/latest/api/auto-updater#windows
[28] https://www.electronjs.org/docs/latest/api/auto-updater
[29] https://web.dev/web-bundles/
[30] https://web.dev/progressively-enhance-your-pwa/
[31] https://developer.microsoft.com/en-us/microsoft-store/pwa/
[32] https://developer.chrome.com/docs/android/trusted-web-activity/quick-start/
[33] https://chromeos.dev/en/publish/pwa-in-play#chrome-os-only

PWABuilder[34] (internally based on the command line tool bubblewrap[35]) for submitting applications. Meanwhile, on other stores such as Apple's App Store, the situation is different and less welcoming, and apps may or may not make it into the App Store,[36] depending on the outcome of the app review. Recently, Oculus, a division of Meta Platforms that produces virtual reality headsets, has announced that PWAs would be accepted into the Oculus Store.[37]

How big a challenge is this? If your users are on one of the platforms whose stores accept PWAs, you can publish your app to the stores in question. Remember linkability as one of the Web's super powers. Your app is discoverable, advertisable, and linkable from the Web, too. Investing in a memorable domain name can sometimes actually be better for discoverability. Even for app stores, people still rely most on recommendations from friends and family members to discover new apps according to Google's research.[38]

5.3 Monetization of apps and in-app content

Apart from making apps themselves available for a fee, apps can also be monetized by selling items as in-app purchases (for example, items in a game app), or by selling subscriptions (for example, video courses in a fitness app). If the developer integrates with payment providers, all of this is available to Web apps as well, but the smooth integration of stores and their related payment systems make this a lot more attractive for platform-specific apps, albeit at a 15–30% commission. For apps built using Trusted Web Activities[39] and delivered through the Google Play Store, developers can now use the Payment Request API[40] and the new Digital Goods API[41] to integrate with Google Play Billing.

How big a challenge is this? When you profit from the convenience of app store billing or in-app purchases, at the same time you also leave a part of your benefits on the table as a commission. As a matter of fact, some apps that are published to app stores even ask their users to make the purchase off-store. One well-known example is Netflix with their external subscriptions.[42]

5.4 Hiring or retraining developers

From personal experience through talking to many of Google's partners, a lot of companies struggle with hiring great Web developers.[43] The talent shortage is real, and recruiting costs are high, which is why startups commonly hire in-house recruiters who often approach recruiting with a breadth-first approach that hasn't helped the reputation of recruiters with IT professionals. Also, companies often already employ teams of Android and/or iOS developers that they cannot just retrain to become Web developers. Creating a PWA requires a high level of specialization that not all Web developers can offer.

How big a challenge is this? In the current economic situation, hiring any kind of developer is difficult. Hiring someone with Web development skills is, in comparison to other platform-specific coding skills, still easier according to StackOverflow surveys,[44] which (in part) also explains the popularity of Web-technologies-based app frameworks such as React Native, Ionic, Flutter, *etc.*

5.5 Legacy apps (and migrating the user base)

It'sn't unusual for companies to have made considerable investments in platform-specific apps, and giving up these investments, as well as a user base acquired over time (not to speak of the vanity install statistics), isn't easy. Apparently, starting from scratch, even when a company has an existing website, appears very unattractive in comparison, but sometimes it does happen.[45]

How big a challenge is this? Vanity is vanity, but once you have set up new, potentially more meaningful, tracking metrics than number of app installs such as increase of indicators of purchase intent,[46] you can start tracking those instead.

5.6 Compatibility with relevant browsers

Web compatibility is still the main issue[47] mentioned in developer surveys like Mozilla's, but also in internal surveys that Google has run. Having to support specific browsers, avoiding or removing a feature that doesn't work across browsers, or making a design look or work the same across browsers are frequently brought up as challenges. Projects such as https://webcompat.com/ collect user-submitted browser bugs and invite interested developers to fix them. Mozilla operates a repository[48] with interventions and patches to enable individual sites to run successfully in Firefox. WebKit maintains a quirks list[49] and was hiring compatibility analysts.[50]

How big a challenge is this? Compatibility is the top priority for Web developers and browser vendors alike. With feature testing and progressive enhancement, impressive apps which behave well on all browsers can be built. For an example of this approach, check out the article Progressively enhance your Progressive Web App.[51]

5.7 Tools and framework support

Apart from browser compatibility, Mozilla's 2020 developer survey[52] likewise showed that developers struggle with tools and frameworks. Supporting multiple frameworks in the same code base, understanding and implementing security measures, plus outdated or inaccurate documentation for frameworks and libraries, and keeping up with the large number of new and existing tools or frameworks were all cited as challenges.

[34] https://www.pwabuilder.com/
[35] https://github.com/GoogleChromeLabs/bubblewrap
[36] https://blog.pwabuilder.com/posts/publish-your-pwa-to-the-ios-app-store/
[37] https://developer.oculus.com/pwa/
[38] https://www.thinkwithgoogle.com/_qs/documents/331/how-users-discover-use-apps-google-research.pdf
[39] https://developer.chrome.com/docs/android/trusted-web-activity/
[40] https://developer.mozilla.org/docs/web/API/Payment_Request_API
[41] https://goo.gle/trusted-web-activity-receive-payments
[42] https://9to5mac.com/2022/07/22/netflix-external-subscription-ios/
[43] https://goo.gle/why-hiring-is-so-hard-in-tech
[44] https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof
[45] https://www.glossy.co/ecommerce/gone-fishin-patagonia-bids-farewell-to-mobile-app/
[46] https://web.dev/betty-crocker/#results
[47] https://insights.developer.mozilla.org/reports/mdn-web-developer-needs-assessment-2020.html#needs-assessment-top-ten-needs
[48] https://github.com/mozilla-extensions/webcompat-addon
[49] https://trac.webkit.org/browser/webkit/trunk/Source/WebCore/page/Quirks.cpp
[50] https://jobs.apple.com/search?search=%22WebKit%20Web%20Compatibility%20Analyst%22&sort=relevance
[51] https://web.dev/progressively-enhance-your-pwa/
[52] https://insights.developer.mozilla.org/reports/mdn-web-developer-needs-assessment-2020.html#needs-assessment-top-ten-needs

How big a challenge is this? The tooling and framework situation in the Web development world is infamous for being confusing and hard to keep up with. In practice, though, companies would use one technology and tooling stack and stay true to it for many years. The world of tech Twitter is one thing, the reality in businesses, where the decades-old jQuery is still (and by a large amount) the most popular framework,[53] is the other.

5.8 Security and certificate pinning

In platform-specific app development, certificate pinning restricts which certificates are considered valid for a particular app. Instead of allowing any trusted certificate to be used, developers pin the certificate authority issuer, public keys, or even end-entity certificates of their choice. Clients connecting to that server will treat all other certificates as invalid and refuse to make an HTTPS connection. The hope is that this renders "person-in-the-middle" attacks impossible, so platform-specific apps are more "secure" than Web apps, where traffic can easily be sniffed with browser DevTools. There are ways to circumvent pinned certificates[54] on all platforms, so it's mostly the theater of security at this point.

How big a challenge is this? As outlined earlier, certificate pinning mostly just increases the effort an attacker has to put into sniffing your traffic and reverse-engineering the functioning of your app; but it doesn't make it impossible.

5.9 Performance limitations

Web applications have seen impressive performance improvements thanks to advanced technologies such as WebAssembly[55] (including SIMD[56]) and general JavaScript engine progress. Nonetheless, a carefully developed, platform-specific app will typically outperform a Web-based application (albeit the situations where this actually matters are limited). With even high-performance audio-editing tools like Soundtrap[57] (thanks to the Web Audio API[58] and AudioWorklet[59]), tools like Jupyter Notebook,[60] and graphics-editing tools like Figma[61] (thanks to WebAssembly), and of course graphics-intensive games like Quake[62] (thanks to WebGL and WebGPU[63] in the future), the boundaries are being pushed at a rapid rate.

How big a challenge is this? There are two types of performance problems: those where truly every frame counts, as in gaming or WebXR experiences, and those where apps feel "janky", or unreliable. For the latter, new APIs such as the View Transitions API[64] can help. For the former, WebGPU is probably the most promising API on the horizon. Rarely a device may just be too slow to render a given experience, which clearly happens with native apps, too, where developers can specify minimum required device capabilities.[65]

[53] https://almanac.httparchive.org/en/2021/javascript#libraries-and-frameworks
[54] https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/
[55] https://webassembly.org/
[56] https://v8.dev/features/simd
[57] https://www.soundtrap.com/
[58] https://developer.mozilla.org/docs/web/API/Web_Audio_API
[59] https://developer.mozilla.org/docs/web/API/AudioWorklet
[60] https://jupyter.org/try
[61] https://www.figma.com/
[62] http://www.quakejs.com/
[63] https://gpuweb.github.io/gpuweb/
[64] https://developer.chrome.com/docs/web-platform/view-transitions/
[65] https://developer.apple.com/support/required-device-capabilities/

6 WHY IS FLUTTER SO POPULAR?

For RQ2, one possible explanation is that it's a Google-backed[66] toolkit for "building beautiful, natively compiled applications for mobile, Web, desktop, and embedded devices from a single codebase". If even Google, as the maker of Android, trusts Flutter enough to build some of its strategic apps with it, such as Stadia[67] (RIP) and Google Ads[68] for both Android and iOS, and Assistant apps[69] on smart display embedded devices, that is quite a signal to send. Also note how Web and desktop are included in Flutter's output options, which means Flutter is no longer limited to just mobile (with submission into app stores as the carrot), and the promise is that it reduces the development cost of apps by the number of targeted platforms. (Prominent target platform omissions so far are Apple CarPlay, WearOS, WatchOS, and tvOS.)

An argument that is frequently brought up for Flutter is hot reloading.[70] On the backend, Flutter also plays well with Firebase,[71] so apps are easy to scale. Important for Web, and as Flutter was initially criticized for rendering everything inaccessibly onto a `<canvas>`, the framework now has two different Web renderers:[72]

HTML renderer: This renderer uses a combination of HTML elements, CSS, canvas elements, and SVG elements, and has a smaller download size.

CanvasKit renderer: This renderer is fully consistent with Flutter mobile and desktop, has faster performance with higher widget density, but adds about 2 MB in download size. By default, Flutter selects the HTML renderer when the app is running in a mobile browser, and the CanvasKit renderer when the app is running in a desktop browser.

Flutter relies on a library of pre-made widgets[73] called Cupertino (for the iOS-native look) and Material (for the Android-native look) that allow developers to quickly develop a good-looking application with a shared code base. It's worth noting that Flutter-built user interfaces are platform-agnostic because Flutter's Skia[74] rendering engine doesn't require any platform-specific UI components. (A downside of this approach of wrapping everything the app needs instead of reusing platform primitives directly is app size.)

Apps in Flutter are developed in Dart, an object-oriented programming language that supports both just-in-time (JIT) and ahead-of-time (AOT) compilation. Flutter compiles directly to native ARM or Intel x64 code, which has a lot of performance advantages. Dart is also easy to pick up for developers coming from any other object-oriented programming language.

Flutter's documentation[75] is generally recognized as best in class and its cookbook application[76] makes getting started with a baseline scaffolding a simple copy and paste job. The Flutter community is thriving and it's easy to find help if you are stuck.

[66] https://docs.flutter.dev/tos
[67] https://stadia.dev/blog/how-flutter-helped-us-make-stadia-controller-setup-better-for-users/
[68] https://flutter.dev/showcase
[69] https://developers.googleblog.com/2019/05/Flutter-io19.html
[70] https://flutter.dev/docs/development/tools/hot-reload
[71] https://firebase.google.com/docs/flutter/setup?platform=ios
[72] https://flutter.dev/docs/development/tools/web-renderers
[73] https://docs.flutter.dev/development/ui/widgets
[74] https://skia.org/
[75] https://flutter.dev/docs
[76] https://docs.flutter.dev/cookbook

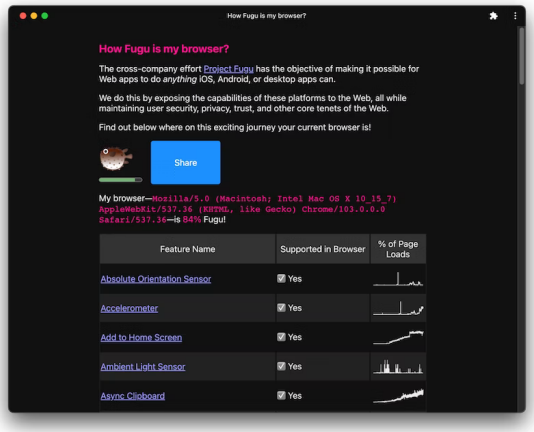

Figure 7: The *How Fugu is my browser?* app

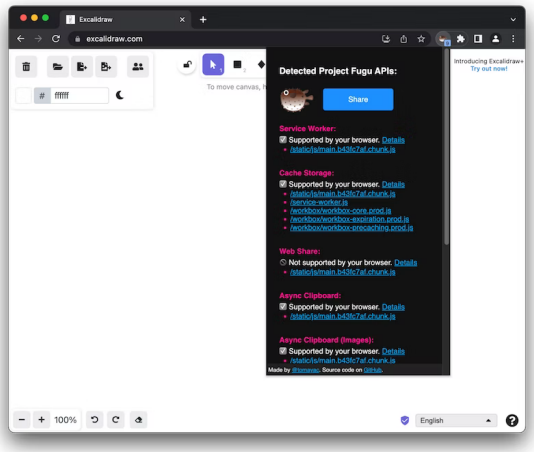

Figure 8: The *How Fugu is the Web?* extension invoked on the site https://excalidraw.com/

7 DISCOVERING THE CAPABLE WEB

Given all the reasons (and counterarguments) listed above why companies currently don't build for the Web, why should you? My hypothesis is that many developers and executives alike don't realize how capable the modern Web has become. Double-clicking an image file so it opens in an associated PWA, making some modifications, saving the changes back to the file and then copying the image contents over into another app or sharing it to an email client is a flow that wasn't possible on the Web before, but which APIs developed in the context of Project Fugu[77] like the File Handling API,[78] the File System Access API,[79] the Async Clipboard API,[80] and the Web Share API[81] have made possible. Project Fugu is a cross company effort to close gaps in the Web's capabilities, enabling new classes of applications to run on the Web. More concretely, this means adding new APIs to browsers that app developers can use to enable previously impossible use cases.

7.1 How Fugu is my browser?

To learn what is possible, check out the application *How Fugu is my browser?*[82] (see Figure 7) and realize what percentage of Project Fugu APIs your browser of choice supports. Not all features are exposed on all platforms—for example, the Contact Picker API[83] is currently only exposed on mobile—so it's technically impossible to reach a score of 100% if you test on desktop (and vice versa on mobile). Therefore, regard this score as a playful competition rather than absolute science. For each tested feature, there's a link to the relevant documentation so you can learn more about the feature. Where feature detection is possible, there is also a note on whether the feature is supported by your browser or not, and finally page load statistics linked to Chrome Statusfeature popularity.[84]

[77] https://developer.chrome.com/capabilities/
[78] https://web.dev/file-handling/
[79] https://web.dev/file-system-access/
[80] https://web.dev/async-clipboard/
[81] https://web.dev/web-share/
[82] https://howfuguismybrowser.dev/
[83] https://w3c.github.io/contact-picker/
[84] https://chromestatus.com/metrics/feature/timeline/popularity

7.2 How Fugu is the Web?

The companion browser extension named *How Fugu is the Web?*[85] (see Figure 8) will help you to find out which Fugu APIs are used by the sites you are visiting. Install this extension from the Chrome Web Store and browse the Web, then notice how the Fugu fish counter on some sites displays a badge with the detected Project Fugu APIs. For example, if you browse to Excalidraw,[86] the counter jumps to 9, since Excalidraw uses nine detectable Project Fugu APIs.

8 CONCLUSIONS

It's undeniable that amazing apps can be built on the Web. Photoshop, VS Code, and Twitter are the stand-out examples in this paper, but there are many others.[87] One of the Web's super powers is its linkability, which is hard to beat on platforms other than the Web. There seems to be a certain tendency for cross-platform app frameworks to become less attractive to developers, with the notable exception of Flutter, which allows for Web as one of its target platforms. Reasons for not building for the Web are easy to find, but it's also not hard to find counter-arguments to take these reasons apart. Some of them rely on outdated or weak assumptions, for example, PWAs not being welcome on app stores, or platform-specific apps being more secure than PWAs. Others are things that are in process, like closing the app gap by adding missing Web platform APIs. Some reasons apply equally to both worlds, for example, for hiring to be a challenge. In this paper, I have given a number of really strong arguments for building for the Web, while also not hiding the fact that the Web is a platform that is still not perfect—yet incredibly capable, as *How Fugu is the Web?* and *How Fugu is my browser?* show—and pointing out that other alternatives exist. And as the three introductory app examples show, the decision is also not mutually exclusive. You can build a powerful Web app, and have a great, platform-specific application at the same time.

[85] https://goo.gle/how-fugu-is-the-web
[86] https://excalidraw.com/
[87] https://developer.chrome.com/fugu-showcase/