blink

By Heidi Adkisson

This summer, a blog post “Safari is the new IE” generated discussion and some measure of angst about the future of web-based applications. Specifically, the author was frustrated by what he saw as Apple’s reluctance to support the latest web technology in Safari. Why would Apple seemingly limit what could be developed for the browser? In theory, to push developers towards developing native iOS apps. Thus, yet another shot was fired in the web app vs. native app discussion.

Which approach (web vs. native) is optimal is not always clear—and in fact the distinction is not even black and white because many so-called native apps have significant web-based components.

Given the recent discussion I thought it was a good time to review the platform possibilities and when they might make the most sense. First, let’s review the three basic options:

- Native Applications
- Web Applications
- Hybrid Applications

Native applications

Native applications are written for a specific hardware platform using a programming language specific to that platform (for example Objective-C for iOS/Mac; Java for Android, C# for Windows). This means you need to develop a separate version of your application for each platform. Native applications have the advantage, however, of (usually) better performance and access to device-level features such as the camera and GPS. They also work when the user is off-line.

A nuance with native apps is the concept of universal apps—apps designed to work on a specific platform (e.g. iOS or Windows 10) but work across all device types on that platform. Apple has made a big push towards universal iOS apps that work across its various iPhones and iPads; Microsoft is doing the same with Windows 10 so that a single app can work across desktop, tablet, and phone devices.

There are also tools that let you code for one platform and port the code to work on another platform. This still requires some additional coding, but it’s less effort than developing two entirely separate apps. Notably, Microsoft provides an iOS-to-Windows 10 port in order to build out more apps for that platform (which have been lacking).

Web apps

In the web app model you create your application using HTML/CSS, which can (in theory) run on any device with a web browser.

You develop and maintain a single code base. However, creating the right experience across a range of device types requires designing and coding responsively—additional effort to make sure the code works well across different form factors and browser types. Responsive design is pretty much de rigueur for websites (Starbucks.com is an exemplar here), but web applications can also be implemented using responsive design.

The modern way of developing web apps is to have much of the functionality run on the client side (minimizing round trips to the server), most commonly using JavaScript. However, this client-side code adds latency because the browsers need to compile JavaScript when loading the page. Thus, web apps can feel more sluggish than their native counterparts.

Hybrid apps

On mobile in particular, apps that would seem to be native (because you downloaded and installed them on your device) are in reality hybrid apps. These are apps that have some functions written natively while others functions are written in HTML that is rendered within the app (rather than in a separate browser). Many e-commerce mobile apps are hybrid apps, with the shopping sections written in HTML but other functions, such as those requiring location awareness, written natively.

Next, let’s look at some platform strategies and when they might make the most sense—for developer and user alike.

Desktop only: Native application

Targeting the desktop with a native application: this is the historical bread-and-butter of application development.

Now this strategy is typically found in three main situations:

- Proprietary line-of-business applications developed internally by a company (such as an application that supports a particular business workflow)
- B-to-B applications particularly enterprise applications such as ERP (Enterprise Resource Planning) systems
- B-to-C applications with specialized functionality such as graphics packages

This strategy can make sense where there is truly no need for a mobile experience—or where high security requirements prevent a mobile experience. However, as more mobile devices find their way into the workplace there is pressure even with traditional line-of-business applications for a mobile experience. People, particularly managers, want mobile access to data and functionality locked in these platform-specific systems. It's created pressure to move these systems to a browser-based experience accessible to a range of device types. Even when mobile isn't a driver, the move away from a native app allows for more centralized management of the application including the deployment of updates.

Desktop only: Web application

Targeting a web application for the desktop only means that you don't have to develop responsively—and you otherwise still get the deployment advantages the web apps bring. In some cases, mostly with decidedly non-mobile line-of-business applications, a single browser type can be targeted, which further streamlines development and maintenance.

There is a bit of history, however, with targeting a specific browser. Some early web applications were IE-specific and required Active X controls (browser plug-ins)—tying the developer's fate to this Microsoft technology. It was a decision many developers lived to regret as IE faded in popularity and Active X was clearly not the direction of the modern web.

Mobile only

Contrasting with the two strategies above is the mobile-only approach. Some experiences are inherently mobile. Instagram out of the starting blocks went with a mobile-only strategy—and for some time it was only offered on a single platform (iOS). JetBlue recently announced that their aircraft mechanics would be using iPad minis for their maintenance work.

In most instances, mobile-only experiences benefit from being native applications.

It is certainly possible to do mobile-only with a web app. However, there are more technical limitations in terms of performance, functionality, and data storage. An intermediate approach is the hybrid app, which leverages web components but can still provide native functionality.

The reality is that in most situations there is a requirement for an app to work across a range of devices and platforms.

In this case, there are three main options:

- A Suite of Native Apps
- A Fully Responsive Web App
- A Web App with Companion Native Mobile App

A suite of native apps

Developing a suite of fully native apps is the most costly strategy from a development standpoint. But a fully native app in most circumstances still provides a superior user experience. Apps representing this strategy include Evernote, 1Password, and Drop Box. These are all designed to be "available anywhere" experiences—data is synced between devices via the cloud. From an experience standpoint cloud syncing can seem almost magical—and it's hard to beat the performance of having data right on your device.

A variant of a "suite of natives" strategy is developing a suite of hybrid apps. In this approach you get the benefits of app store distribution and access to device-level functions, but the development among apps can be streamlined because they can share web-based components.

Native apps require, in many instances, the user to initiate upgrades. This allows the user to upgrade at his or her own pace, but can also result in laggards running outdated versions.

A fully responsive web app

The frustration expressed in "Safari is the New IE" points to a current constraint with web app development. Depending upon what your app needs to do it can be difficult achieving parity with the native experience. Fully responsive design requires that the application behaves in a way that doesn't degrade the experience across the range of device types. The "mobile first" methodology was intended to address the challenge of designing responsively—specifically to mitigate the risk that the desktop experience would simply be squeezed into a mobile form factor. However, mobile first may not make sense in all contexts, particularly when the mobile experience is truly secondary. Responsive design is inherently a balancing act.

For commercial software, going with a pure web app approach also means foregoing app store distribution; lacking app store visibility can make it more difficult for users to discover your app. The flip side, however, of bypassing the app store is that you don't have to submit you app to the store's approval process, you don't have to pay a percentage of the purchase price to the store, and you have the potential to have more direct communication with your customers. Another plus—applicable to both commercial and internally-developed web apps—is that you have a single code base to manage and you don't need to hire a staff of developers with platform-specific coding skills.

A web app with companion native app for mobile

Creating a web app—with some degree of responsive behavior—along with a companion native mobile app is a common strategy. Native mobile apps give you the opportunity to leverage features such as the camera and location-awareness—features that aren't suited to the desktop. You've likely experienced this strategy with online banking. You might review your account activity on the desktop. Then, need to deposit a check? Whip out the app, take a photo of the check, and voilà.

The wrap-up

This has been a decided fly-by on the topic of platform strategy—the decision ultimately includes other nuances not covered here. A good start, however, is always the user experience. What are the likely contexts of use? What devices and platforms are likely to be used in that context? Are some tasks inherently more suited to one context over another? How could uniquely mobile experiences add to what you can deliver? And of course, we are here to help organizations through these questions—just give us a ring.

> *Heidi works in Interaction Design and is a Partner at Blink. She divides her leisure time between classical music, cooking, and the Seattle Mariners.*