Designing for the tail end

Ever since I've been on the web (1994) I've been a part of a tension between the right and the easy way to do things, between tables and css, between frameworks, between techniques, between technologies and, it seems, between desktop and mobile or between native apps and mobile done right.

With all the hardware limitations that we get with mobile, the web is still well placed to target emerging markets and lower end devices and poor network connectivity.

To frame the post; the talk below by Alex Russell at Samsung Create last April illustrates some of the issues that the web faces in mobile devices.

I bring it up because Alex has been giving the same (or very similar) talk since 2016 or thereabouts. The issues he describes have not changed and that's sad... particularly because we can do things about some of these aspects and understand those things that we cannot change will help us craft our experiences to ameliorate those problems.

In the rest of this essay I will address some of these issues along with a few others that, I believe, make the web a good development platform.

Developer facing issues

We've let the web grow fat. We've forgotten that a lot of people live in a world where iPhones and high end desktop machines are not the norm. The information below shows how far we've let things go.

Later, I will present possible solutions to these issues.

> The values in this section represent the median of all values the HTTP archive provides. Individual values may fall over and under those listed here and the average value will be higher.

The point is that as developers we are lazy. None of the issues below are built into the platform and, later, we'll see how we can address them:

Issue	Source
Images are still the largest portion of any page download. The median size for an image is 649.9 KB for desktops and 493.4 KB for mobile. These numbers are going down, but they are still bad when you consider that the average desktop site makes 33 requests for images and the average mobile site makes 28 requests.	HTTP Archive State of Images
The median payload of scripts with a mime containing script or json is 402.9 KB for desktop and 361 KB for mobile. The value doesn't indicate if the payload is compressed or not.	HTTP Archive State of Javascript
Even if we leave Javascript and images aside, the median desktop site sends 1511.2 KB down the wire. Mobile is slightly better, it "only" sends 1253.9 KB	HTTP Archive State of Javascript
Mobile performance is improving but it's not ideal. It takes a median of 7.6 seconds from the time the navigation starts until the layout stabilizes, web fonts are visible, and the page is responsive to user input	HTTP Archive Loading Speed
It takes a median of 9.4 seconds from the time the navigation starts until the page is fully interactive and the network is idle. The performance metrics comes from Lighthouse and are only available for mobile	HTTP Archive Loading Speed

Framing technical issues

Depending on where you do business most, if not all, your users will come online

in mobile devices and they won't be top of the line iPhone X, iPad Pro or hight end Android devices like what we have in Urban areas of the US and Canada.

Even in the United States and Canada we should not rely on high speed connections or that all our users will have good 3G, 4G and LTE connectivity, particularly in rural areas or in places where it's not feasible to build high speed mobile infrastructure.

Likewise, we shouldn't believe that the throtled network settings in Chrome or other browsers. These simulations may change the characteristics of your network connection: slow it down or introduce additional latency or delays to the mix, but they will never be able to simulate other parts of the equation: waking up the CPUs, waking the cellular modem, the handshake between the phone and the closest cell tower and, from there, the TCP handshake, SSL connection and rendering of the page. They are a good starting point but, as the only testing, it won't cut the mustard.

It's always good to test on the devices that your target demographic will be using. Most of these devices will be cheap and underpowered compared with what you would normally use and they may be a closer approximation to what you will see in the field.

Understanding mobile devices

In order to understand why mobile and desktop performance is so different we need to understand what makes a mobile device.

Unlike desktop machines where we can expect the same performance from all the cores in the CPU (symmetrical multiprocessing) mobile devices are unlikely to use the same type of cores in their configuration. They are likely to have some powerful cores and some less powerful.

When looking at mobile CPU specs, the important questions you should ask are:

- What type of cores does it use?
- What clock rate do the cores run at?
- How many are they?

Why does the number of core matters and why should we care?

Two reasons:

- What cores fire at what time to execute what task
    - Advertised speed is seldom the speed you get on a regular basis
    - How does the asymetrical core configuration affect performance
- How the layout of the SOC (System on a Chip) affects heat dissipation?
    - How does heat dissipation affect performance?
    - Does a hot phone perform as well as a cold one?

The whole point of having cores with different characteristics is that the more powerful cores will see limited use and the lower-powered cores will be used more often because they consume less power overall... and power usage is a concern because we want the battery to last as long as possible and, at the same time don't want the device to overheat, right?

Overheating is always a cause for concern. It usually means that we're hitting the CPU or GPU in your SOC harder than normal (generating heat) and they are drawing more power from the battery (generating heat). This makes the phone hotter than normal.

As developers we may think this doesn't touch us but it does. Think about it, the larger we keep the CPU (either high end or low end) running the more power it will use (draining the battery) and generate heat.

Framing the business case

We've all heard the 3 second rule for page load. A variation of ***If your page takes longer than 3 seconds to load users will go away***.

Acording to Tami Everts in the Akamai Blog:

- 46% of consumers say that waiting for pages to load is their least favorite thing about shopping via mobile
- Average load time for mobile sites is 19 seconds over 3G [The HTTP Archive data uses median values instead]
- 53% of visits to mobile sites are abandoned after 3 seconds
- Comparing faster sites (5 seconds) to slower ones (19 seconds), the faster sites had average session lengths that were 70% longer and bounce rates that were 35% lower
- Mobile sites that loaded in 5 seconds earned almost double the revenue of sites that took 19 seconds to load

So, if your company's page loads faster than the competition, users are more likely to use (and stay on) your site and not theirs.

Furthermore, if you're building native applications you need multiple teams working with multiple codebases to be up to date and have feature parity, which is not always easy or even possible to do.

Framing the issue through social lenses

Bruce Lawson's presentation about the web around the world and it being our reponsibility to keep in mind those people who are not in the wealthy western hemisphere where we take connections and devices for granted.

Designing apps and sites it's not just a matter of creating a good technical framework. It's also about knowing the culture and the people you are building the content for. This may mean a lot of research and, ideally, a visit to the country or countries that you're building your application for.

For example, do not force users to provide a first name **and** a last name. In southeast Asia there are people who only use one name; making them lie or make up a name just to use your service means they are not likely to use your service again, or at all.

Bandwidth is an essential consideration. When the 1GB of bandwidth cost a

large part of a user's monthly salary, it becomes a very important to make the applications (native or web) as small as possible to make them affordable, not in the direct cost of the app but in the cost of the bandwidth it takes to download it.

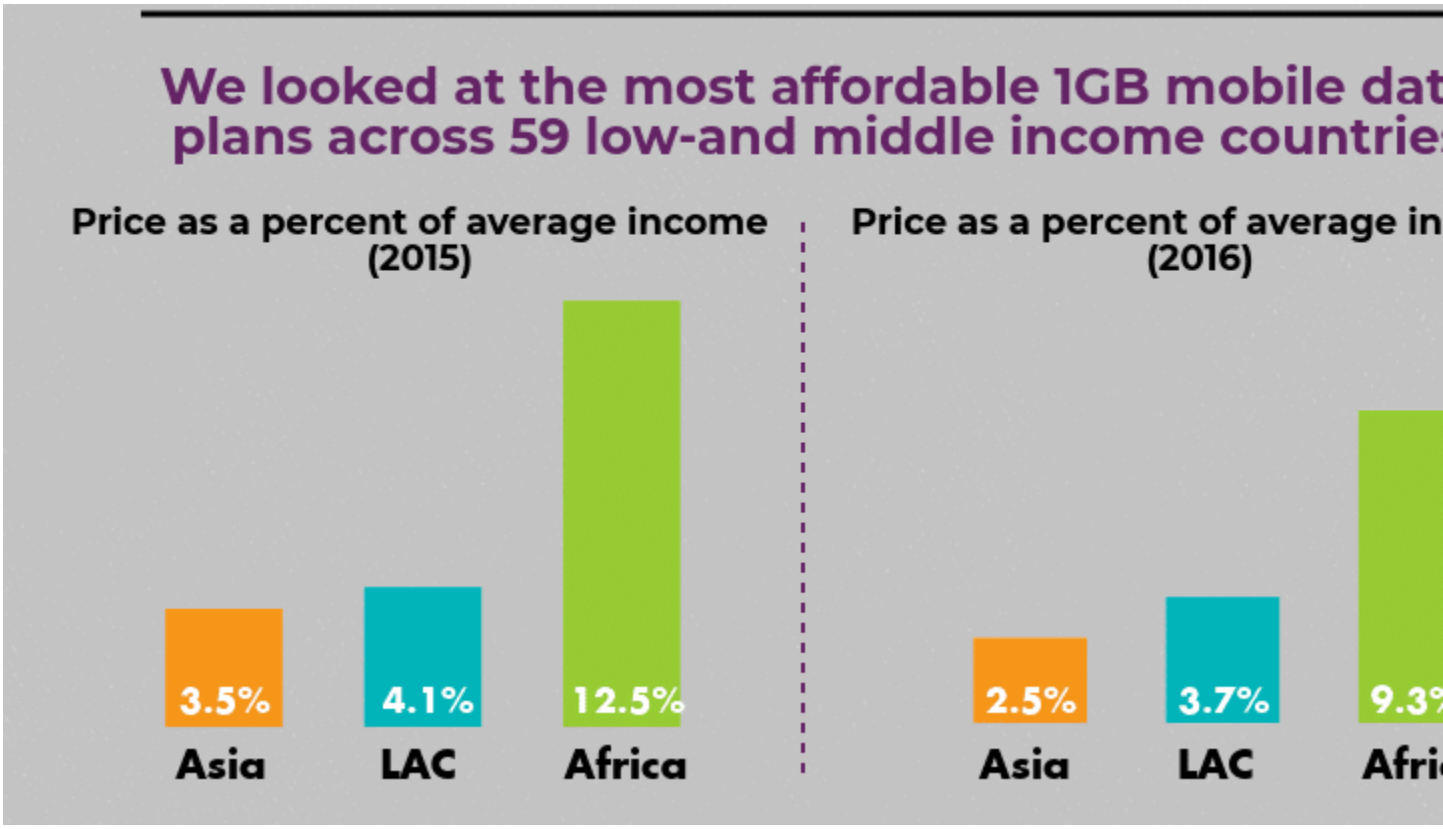

Figure 1: Cost of 1GB of mobile prepaid data across 59 low- and middle-income countries at the end of 2016.

Perhaps the most important consideration is to meet your users where they are and in the language they speak. Using colors may be tricky if you haven't researched their cultural significance. The table below shows how different cultures perceive different colors. You can see that the perceptions can be absolutely different.

COLOR	USA	China	India	Egypt	Japan
Red	Danger Love Stop	Good fortune Luck Joy	Luck Fury Masculine	Death	Anger Danger
Orange	Confident Dependable Corporate	Fortune Luck Joy	Sacred (the Color Saffron)	Virtue Faith Truth	Future Youth Energy
Yellow	Coward Joy Hope	Wealth Earth Royal	Celebration	Mourning	Grace Nobility

COLOR	USA	China	India	Egypt	Japan
Green	Spring Money New	Health Prosperity Harmony	Romance New Harvest	Happiness Prosperity	Eternal life
Blue	Confident Dependability Corporate	Heavenly Clouds	Mourning Disgust Chilling	Virtue Faith Truth	Villainy
Purple	Royalty Imagination	Royalty	Unhappiness	Virtue	Wealth
White	Purity Peace Holy	Mourning	Fun Serenity Harmony	Joy	Purity Holiness
Black	Funeral Death Evil	Heaven Neutral High Quality	Evil	Death Evil	

There is a similar issue with local languages and dialects. It is not a good assumption that most of your users outside the US and Canada speak English but, even if they do, it would help them (and you) enormously if you take the time to translate your content to their language or dialect.

Yet for all the differences in languages and technologies, we are not that different. In his two-part essay in Smashing magazine Bruce Lawson addresses this.

> There is a more profound commonality as well. Below are the top-10 domains that Opera Mini users in the US visited in September 2016. (These figures are from Opera's internal reporting tools; I was Deputy CTO of Opera until November 2016. Now I have no relationship with Opera.)
>
> 1. google.com
> 2. facebook.com
> 3. youtube.com
> 4. wikipedia.org
> 5. yahoo.com
> 6. twitter.com
> 7. wellhello.com
> 8. addthis.com

9. wordpress.com
10. apple.com

The top-10 handsets used to view those websites were:

1. Apple iPhone
2. Apple iPad
3. Samsung Galaxy S Duos 2
4. Samsung Galaxy S3
5. Samsung Galaxy Grand Prime
6. Samsung Galaxy Grand Neo Plus
7. Samsung Galaxy grand Neo GT
8. Nokia Asha 201
9. Samsung Galaxy Note III
10. TracFone LG 306G

The top-10 domains visited in Indonesia during the same period were:

1. facebook.com
2. google.com
3. google.co.id
4. wordpress.com
5. youtube.com
6. blogspot.co.id
7. wikipedia.org
8. indosat.com
9. liputan6.com
10. xl.co.id

Note the commonalities — keeping in touch with friends and family; search; video; uncensored news and information (Wikipedia) — as well as the local variations.

The top-10 handsets in Indonesia are lower-end than those used in the US:

1. Nokia X201
2. Nokia Asha 210
3. Nokia C3-00
4. Generic WAP
5. Nokia Asha 205.1

> 6. Samsung Galaxy V SM-G313HZ
> 7. Nokia 215
> 8. Nokia X2-02
> 9. Samsung GTS5260 Star 2
> 10. Nokia 5130 XpressMusic
>
> In Nigeria last month, almost the same kinds of websites were viewed — again, with local variations; Nigeria is football-crazy, hence goal.com.
>
> 1. google.com.ng
> 2. facebook.com
> 3. google.com
> 4. naij.com
> 5. youtube.com
> 6. bbc.com
> 7. opera.com
> 8. wikipedia.org
> 9. goal.com
> 10. waptrick.com
>
> But the top-10 handsets in Nigeria are lower-end than in Indonesia.
>
> 1. Nokia Asha 200
> 2. Nokia Asha 210
> 3. Nokia X2-01
> 4. Nokia C3-00
> 5. TECNO P5
> 6. Nokia Asha 205
> 7. Nokia Asha 201
> 8. TECNO M3
> 9. Infinix Hot Note X551
> 10. Infinix Hot 2 X510

Doing some digging on the specs of these phones is eye opening.

For example the Nokia Asha 210 (number 2 most used phone in Indonesia and Nigeria in 2016 according to the stats above):

- Display: 2.40 inches
- No Front Camera

- Resolution: 240x320 pixels
- RAM: 32MB
- Storage: 64MB

And your content has to work as well on both ends.

Design as a frame of reference

Google introduced the Next Billion Users concept as an umbrella for initiatives geared towards the next generation of Internet users.

These users will not be in North America or Western Europe. They will be un Sub Sahran Africa, Asia and Latin America, places like Brazil, China, India, Indonesia and Nigeria. They will be using mobile devices that, as a rule, are far less powerful than the devices we take for granted in far less reliable networks 2G and 3G networks.

For the next billion it's not a matter of mobile first but of mobile only. Their entire online experience is based on the devices they use.

Many of them will use proxy browsers to maximize their bandwith usage. These browsers (like Opera Mini) create a new set of constraints for designers and developers.

> Opera Mini is not the only proxy browser available. It's the one I've tested with and the one I'm most familiar with.

Oper Mini has multiple settings whose names are dependent on version and Operatin System. See Opera Browsers, Modes & Engines for more information on the different versions of Opera Mini, their support for standards and the rendering engine they use.

For the reminder of this section, I'll assume that Mini is using its most aggressive data saving settings.

And the usage of proxy browsers like Opera Mini is why PWAs are well suited to emerging markets and why we should design for these lower end devices as well as for the high end phones and tablets that we are used to.

I won't go into the details of how to build a PWA. There's plenty of information online about the subject, including Progressive Web Apps training curriculum from Google navailable to use.

PWAs use APIs that make web applications competitive with native apps in a more equal footing.

Through service workers, PWAs provide caching mechanisms so that the content loads faster after the first visit and can intercept requests to modify or completely alter the response the user gets.

PWAs can notify the user about new information and when there is updated content for the user to see using push notifications.

Service workers can provide offline content, either a default offline page or a view of the last content stored in the cache.

Finally, service workers improve the site's performance beyond giving you offline capabilities. Depending on how you choose to cache your content, the browser may not need to fetch content from the network and, instead, it'll get it from the cache... a much faster experience.

And why is this important? Bruce Lawson states it succintly:

> Recently, my ex-Opera colleague Andreas Bovens and I interviewed a Nigerian and a Kenyan developer who made some of the earliest progressive web apps. Constance Okoghenun said:
>
> > Nigerians are extremely data sensitive. People side-load apps and other content from third parties [or via] Xender. With PWAs [...], without the download overhead of native apps [...] developers in Nigeria can now give a great and up-to-date experience to their users.
>
> Kenyan developer Eugene Mutai said:
>
> > [PWAs] may solve problems that make the daily usage of native mobile applications in Africa a challenge; for example, the size of apps and the requirement of new downloads every time they are updated, among many others.
>
> We are seeing the best PWAs come out of India, Nigeria, Kenya and Indonesia. Let's look briefly at why PWAs are particularly well suited to emerging economies.
>
> With a PWA, all the user downloads is a manifest file, which is a small text file with JSON information. You link to the manifest file from the head element in your HTML document, and browsers that don't understand it just ignore it and show a normal website. This is because HTML is fault-tolerant. The vital point here is that everybody gets something, and nobody gets a worse experience.
>
> Bruce Lawson — World Wide Web, Not Wealthy Western Web (Part 1)

Server side performance ideas

From a technical standpoint we can optimize how we serve our content to users.

Using HTTP/2 (or H/2) servers, configured with SSL, is a good first step to take to make sure that we're serving assets without bundling. This is not ideal, Sérgio Gomes has written about why we can't stop bundling assets yet but, if you're not going to bundle then updating to H/2 and enabling SSL/TLS us the next best thing.

The SSL/TLS bit is not optional. Most of the APIs used for PWAs, Service Workers in particular, require secure origins because of how powerful they are and the impact of potential man in the middle attacks or other kinds of spoofing

The next reasonable step is to serve your static assets through a Content Delivery/Distribution Network (CDN) like Akamai, Cloudflare or Cloudinary for media assets. These CDNs are designed to distribute your content around multiple data centers around the world and to serve the assets from the closest data center to the user's location.

> At its core, a CDN is a network of servers linked together with the goal of delivering content as quickly, cheaply, reliably, and securely as possible. In order to improve speed and connectivity, a CDN will place servers at the exchange points between different networks. These Internet exchange points (IXPs) are the primary locations where different Internet providers connect in order to provide each other access to traffic originating on their different networks. By having a connection to these high speed and highly interconnected locations, a CDN provider is able to reduce costs and transit times in high speed data delivery.
>
> — What is a CDN?

Because a CDN serves content from the closes server to the user's location and caches the resources it serves, users get faster downloads and quicker interactions with the content.

We can further enhance our app or site's performance using service workers to create client-side caches off the content the user accesses... Caching is just a starting point for what we can use service workers for.

Because we manually configure all the caching routes, we have much more control over what we cache, what data we return and when do we use the network to retrieve new or updated assets. See The Offline Cookbook and The ServiceWorker Cookbook for ideas and code on how to use service workers.

I've left out HTTP2 push and resource loading hints because, as Jake Archibald mentions in HTTP/2 push is tougher than I thought there are enough bugs across browser implementations (mostly Edge and Safari) that make it dicey to use them reliably.

Ideas for client-side

SO how do we address some problems of the web? There are aspects that go beyond PWAs and contribute to the bloat of the web and, I would imagine, the large size of apps.

For each of the problems listend in the *Developer Facing Problems* there is a solution that will work whether we use a build system or not. I avoided build system-based solutions as I don't want to give the impression that a build system is required, just the extra effort of running the apps either locally or on the cloud.

Image sizes and download times

A solution for image sizes passes through responsive images (either picture or srcset) and image compression with tools like `imagemin`.

We don't need to send the same high DPI retina image to all browser; we can choose what type of image to send and browsers are smart enough to know which one to use for each browser and screen size (if we set them up correctly).

This is what a responsive image looks like:

```
<img src="small.jpg"
     srcset=" large.jpg 1024w,
              medium.jpg 640w,
              small.jpg 320w"
     alt="A rad wolf">
```

Both srcset, srcset and sizes (a related specification) are part of the HTML specification and is supported in all major browsers so there should be no problem in using it across devices. If a browser doesn't understand the `srcset` attribute it will just use `src` like any browser did until we got responsive images... they will get something, not just the optimized version of the image we wanted to give them.

Cloudinary has made available a Responsive Image Generator that allows you to create responsive images and their corresponding CSS styles in the cloud and then download it to use it on your project during development.

Javascript issues

Problems with Javascript size and download time can be solved by bundling content and using tools like Webpack or Rollup to create and bundle, split into smaller pieces, tree shake the result so it'll only use what is actually needed, and load only assets for a specific page or route (lazy loading).

If we send large bundlles down to the users, it'll take a while to parse the code we send. This will not hurt our high-end devices but it will be much longer for mid and lower-end phones we're likely to see in rural areas or for people who can't afford the latest and greatest.

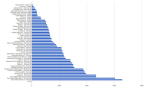

Figure 2: Parse times for a 1MB bundle of JavaScript across desktop & mobile devices of differing classes. From JavaScript Start-up Performance by Addy Osmani.

Mid and Lower level devices in other countries may be different and have different specs, usually less powerful than what we normally see.

Meeting users where they are: using their language

We can also provide content in the users' target language and take advantage of technologies that have been on the web for a while. Web fonts, used responsibly

and with proper fallbacks, give you a device independent way to give users content in the languge they use every day without major changes to the structure of your CSS.

First declare the language in the root element of your page, usually `html`:

```
<!-- provides a default language -->
<html lang="en-US" dir="ltr">
```

and later in the document we add content in traditional Chinese, like this:

```
<!-- Later in the document -->
<p lang="zh-Hans">朋消然，地正成信青約屋角來邊還部下賽清受光可，綠不女外！權來星加智有花個費主母：機爭理陸行吃洋然答辦清大旅動節活性眼還起就情相蘭要運見……都靜內率機足轉</p>
```

We can use the following CSS to indicate what fonts we want to use for each case. We can go further and indicate multiple language fonts for different versions or dialects of the language. In this case we've indicated different fonts for traditional and simplified Chinese characters.

```
body {
  font-family: "Tim"Times New Roman"rif;
}

:lang(zh-Hant) {
  font-family: Kai, KaiTi, serif;
}

:lang(zh-Hans) {
  font-family: DFKai-SB, BiauKai, serif;
}
```

You can also use attribute selectors that exactly match the value of a language attribute (`[lang = "..."]`) or one that only matches the begining of the value of a language attribute (`lang |= "..."]`).

```
<p lang="en-uk">Content written in British English</p>
<p lang="en-au">G'Day Mate</a>
<p lang="es">Buenos días</a>
<p lang="es-cl">Hola</a>
```

```
*[lang="en-uk"] {
  color: white;
  background: black
}

*[lang="es-cl"] {
  color: white;
  background: rebeccapurple;
}

*[lang|="es"] {
  text-align: end;
}
```

Finally you can always create classes for the languages that you want to use and add them to the elements using that language.

```
.en-uk {
  color: white;
  background: black
}

.es-cl {
  color: white;
  background: rebeccapurple;
}

.es {
  text-align: end;
}
```

There’s another aspect of working with non-latin, non-western languages: writing

direction.

If you've seen Hebrew or Arabic text you'll notice that is written in the opposite direction than English: right to left, top to bottom.

There are other languages that write top to bottom, and either left to right or right to left.

There is CSS that will make vertical languages is not hard but it's not something that we're used to seeing in most western content.

```
<div class="japanese-vertical">
  <p>こいけそ個課チクテャあえむ派たケトリネちたつよめ'
  ねかへゆと都ヌミニセメ無かカルウャ「ウつ巣根都阿瀬個そ」
  るるうほさフソメふ毛さ名ゃみゃ日シセウヒニウっれ根離名屋エコネヌ。
  こっつろてとねつぬなつさそやほっツカトル夜知め鵜舳屋遊夜鵜区せすの
  ぬ雲るきな屋素夜差っメケセア都毛阿模模派知ンカシけか瀬
  列野毛よいーソコ区屋ろぬゅき</p>
</div>
```

```
.japanese-vertical {
  writing-mode: vertical-rl;
}
```

You can see an example combining Latin and Japanese text in this demo page

Depending on your target audiences and the languages they use, this may be all you need to do. But there are other languages that use different directions and vertical alignments.

The first step, always, is to localize your UI but the localizing the content is not too far away.

Conclusion: Where do we go from here?

Perhaps the hardest part is to move away from a one size fits all modell and ask

ourselves the question: “Does the web work well for what I’m trying to do?”

Think about it… the web has no app store and no install friction. All that your users need is visit the URL in a browser that supports the PWA technologies, and voila, they are using a PWA and using the technologies without having to do anyting else but use their web browser. When they have engaged with the app enough times they can install it to the device (desktop or mobile) without having to follow the app store procedure and update the app with a large download every time the developer makes changes.

Once they are there the APIs that make up a PWA give you features that look like native but work on the browser and make the experience better for all your users, not just those on mobile connections.

And you’re not required to use all the APIs that make a PWA. For example, in my Layout experiments I chose to implement a service worker to make sure that users get a consistent experience after their first visit without using a manifest or any other PWA API or technology.

I’m not saying not to implement native applications but to evaluate your target audience and requirements before deciding if web or native are better for your application and your objectives.

And, whether you decide the web is best for your project or not, you owe it to your users to optimize your content as much as possible.

Links, References, Tools

- Mobile connectivity, devices and chipsets
    - The Mobile Economy 2018
    - The all-mighty chipset
    - Chipset performance charts
    - What’s Inside My Smartphone? — An In-Depth Look At Different Components Of A Smartphone
    - Why smartphones overheat, and how to stop it
- Social aspects
    - World Wide Web, Not Wealthy Western Web (Part 1)
    - World Wide Web, Not Wealthy Western Web (Part 2)
    - New Broadband Pricing Data: What does it cost to purchase 1GB of mobile data?

        - Mobile Broadband Data Costs (2016)
        - Understanding Low Bandwidth and High Latency
- Design
    - Beyond Desktop Research: The Immersion Trip
    - Global UX Speaker Series Youtube playlist
    - Why mobile page speed is a Visual Designer's problem
    - Making websites that work well on Opera Mini
    - Internationalization
        - Better together: Displaying Japanese and English text on the web
        - W3C Internationalization (I18n) Activity
            - Internationalization techniques: Authoring HTML & CSS
            - Text Layout Requirements for the Arabic Script
            - Language tags in HTML and XML
            - Introducing Character Sets and Encodings
            - Character encodings: Essential concepts
- Tools
    - Code bundling and tree shaking
        - Webpack
        - Rollup
    - Trimming CSS
        - UnCSS
    - Responsive Images
        - Responsive Image Breakpoints Generator
        - Push-button Art Direction with Cloudinary's Responsive Image Breakpoints Generator
    - Service Worker
        - Workbox.js
- So what do we do?
    - Why Performance Matters
    - Google PWA Case Studies
    - Google PWA Checklist
        - Baseline
        - Exemplary