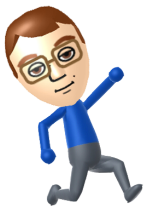
Master's Thesis

The advent of the web through Progressive Web Apps.

How Progressive Web Apps can revolutionize application development ?



Alexis THOREL - *5A IW2* | Hervé TUTUAKU - *5A IW2*

Master's Thesis master : Gael Coat

5A 2016/2017

Introduction

Easily referred as one, if not, the most considerable advance in technology of the past twenty-century, Internet, which first was a way to interconnect computers together and transfer small data from a machine to another on a network, have now dramatically changed but has kept the same fundamental basis.
This is on those basis that was created what we now call the Web, which alike, Internet, have incredibly grown up and changed since it was created in the sixties. The Web platform has constantly evolved through the years.

As the Human specie, which have shown an extraordinary skill at adapting itself to its environment and learn new skills, the Web is an adaptive platform. In contradiction with specified platform such as the Minitel. Often seen as the precursor of Internet, the Minitel did failed to adapt and disappeared.

There are trade offs on the spectrum of generalist and specified, and one isn’t necessarily better than the other. Specified platform can address a strong niche that others aren’t fighting for, they can do well, but are beholden to that niche.

Flexible platforms has something that others don’t, they are able to detect changes in the surrounding landscape and evolve in consequence by borrowing the best ideas from other platforms and finding a solution for the capabilities to adjust to themselves.

The Web has evolved many times before, and yet, it has still a lot to do to become a more reliable platform.

The Web was initially perceived as a document platform. Being able to publish and share documents that could connect to each other anywhere was straightforward and incredible. This ability unfettered the sharing of the world information at a level that has never been seen. To allow a richer semantics, new elements were added to existing ones. CSS brought much richer styling. And then when added to the platform, the scriptability via JavaScript brought the Web in a whole other dimension. Until then creating dynamic content was made by sending requests down from the server via CGI and other protocols.

Then the Web evolve in the desktop application platform. Thanks to technologies like XMLHttpRequest, which we will discuss later. The platform finally had a way to do round trips to the server that wasn’t as limited as frames or CGI pushing data. But despite the helpful APIs that provide   XHR, it was not trivial to build application over the Web.

And this frustration led to the point where developers start doubting about the Web.

Why had there been so many tensions about contorting this document platform while we had native applications which were able to have an access to the full device power ?

The frictionless distribution model that are URLs was the reason.

Native applications were more reliable because it's took advantage of a more powerful platform at the time, where Web applications just couldn't compete.

For instance, applications like Microsoft MapPoint, were only possible on the native platform thanks to its distribution model, there were no need to download the map over and over as we move on it. All the necessary data were backed-in the floppy disk. Even though we finally had the XHR technology to go fetch the data while moving on the screen, the Internet connexion speed was not able to keep up, we had to wait while loading new portion of the map pixels per pixels.

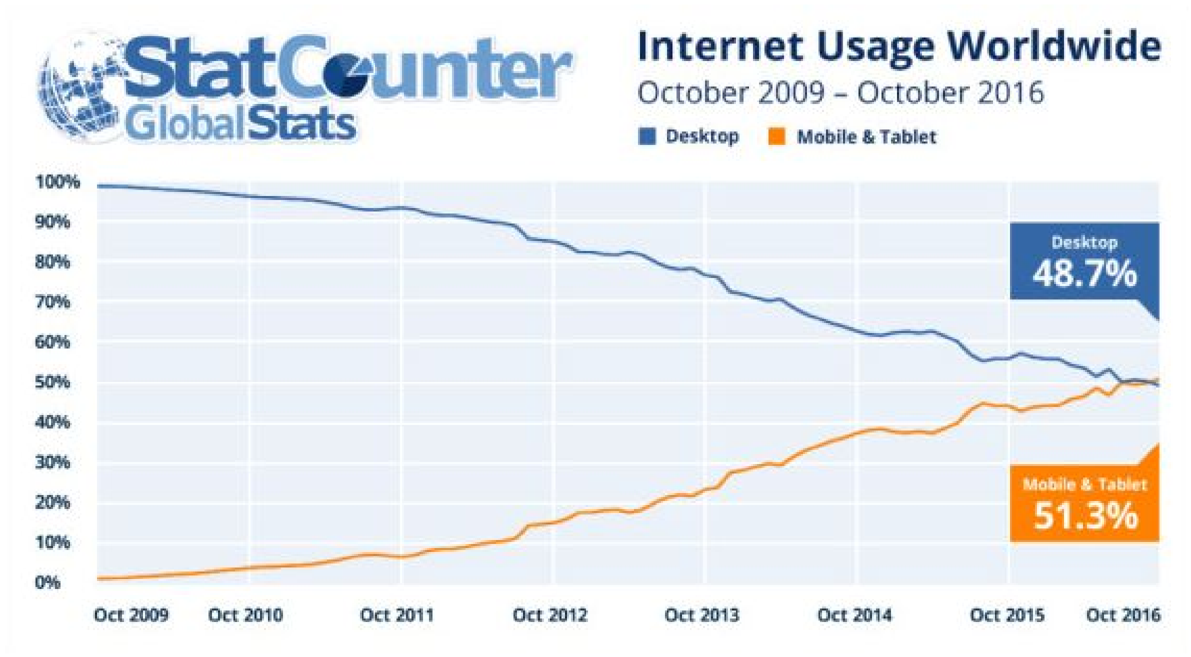
A trade off were made here, as the data were on the floppy, the map could not be up to date, but it promise the ability to the user to use it flawlessly.

Until the Internet connexion speed rise up in developed country, the Web was unable to keep this promise.

But later, the network improvements did give to the Web the ability to express itself at its full power. Application like *Google Maps*, showed how powerful the low friction URL distribution model is. Allowing everyone to find and share localization through a single string of character. Making it universal with no need to download or buy any additional software. Everything lives on the Web and promise to up to date and fast.

Nevertheless, this promise is kept only if we live in a developed country where wired Internet reach gigabytes and browsers run on powerful desktop CPU, because in developing country where internet connection pains to reach megabytes and where the average population use low end device, this promise is totally broken.

As said before, the Web evolved, and our ways to consume it too. More than ever the mobile is the future established way to reach informations.



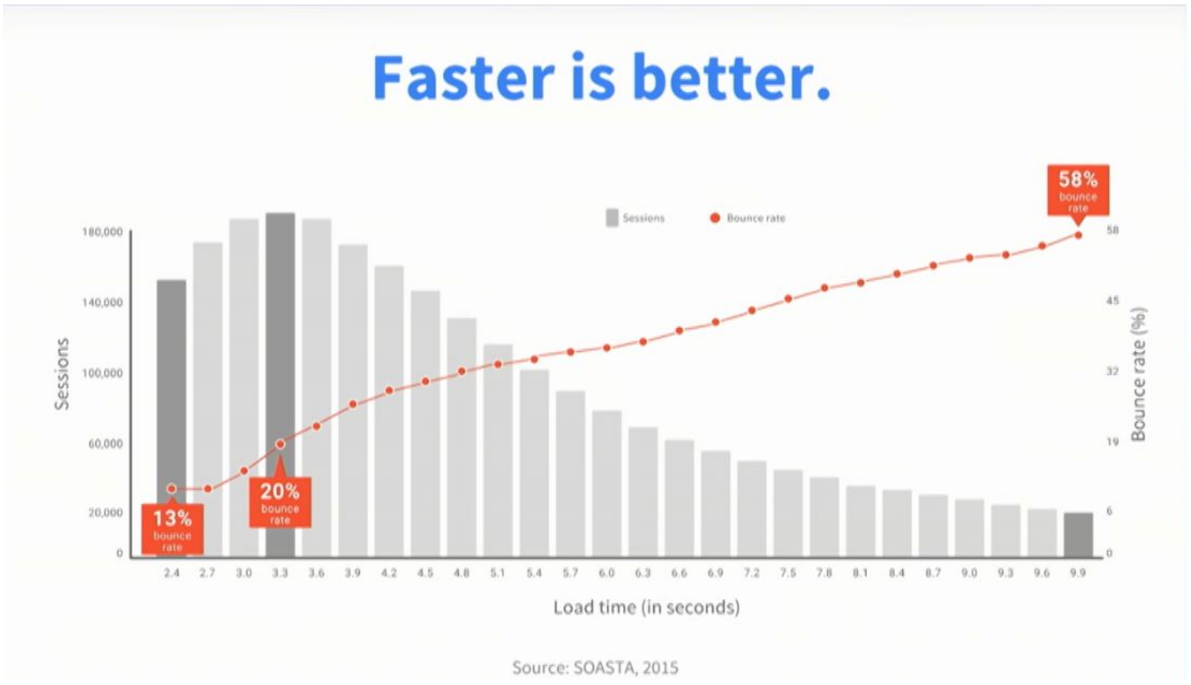
[1]

For the first time, the Internet use on mobile exceed the usage on desktop, respectively 51.3% to 48.7%, which confirm that the web need to evolve once again to adapt to this new transformation of consuming.

As the usages moved, from powerful high speed desktop computers to mobile devices. The way to develop websites has not adapted, or when it try to do so, it’s sometimes in the wrong way.

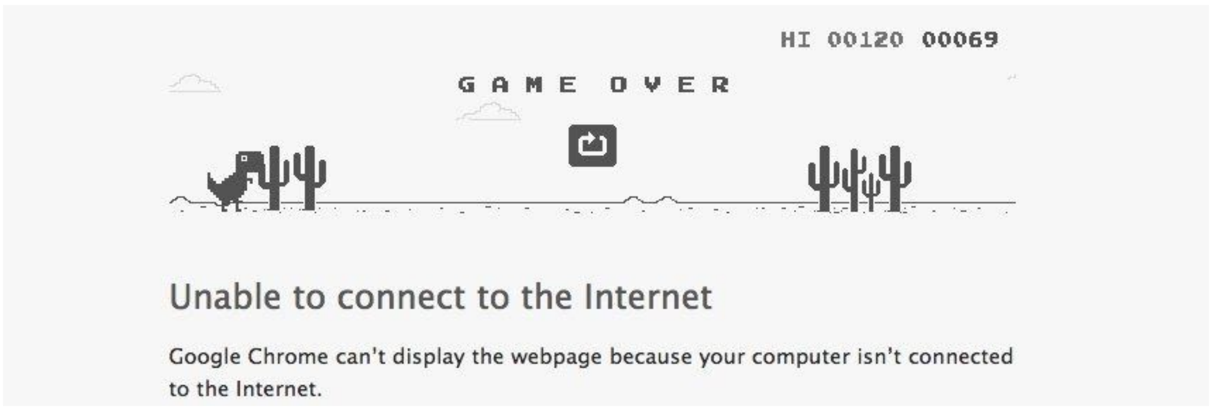
The web page bloat crisis that the web have to face is a serious break to its transformation to mobile.

[1] Source : http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide



[2]

This failure to quickly adapt the the mobile led to the broad development of the native applications on mobile. As the graphic demonstrate, the bounce rate significatively increase as the page load time does. Leading the users to switch from the web platform to a more reliable one, as native apps are better at caching contents or gracefully degrade without blaming the user to be offline.

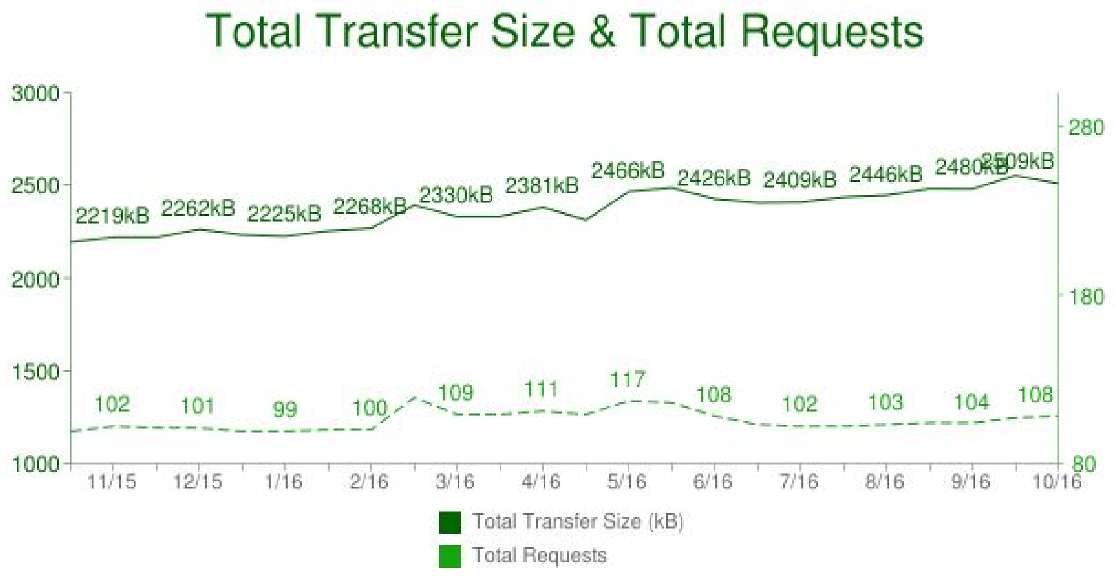


Unable to connect to the Internet

Google Chrome can't display the webpage because your computer isn't connected to the Internet.

As the scripting enhance the experience on desktop, via JavaScript frameworks and libraries. The cost of loading those libraries on a mobile network which is more inconsistent than wired, is way too high.

[2] Source : Alex Russel - https://youtu.be/9Jef9IluQw0?t=10m37s



[3]

The size of average web page keep increasing, since 2014 it has increased by more 15%[4].

To try to solve this crisis in the right way, web developers has comes to a solution. **The Progressive Web Apps.**

The idea of PWA first comes from progressive enhancement. Progressive enhancement and graceful degradation have the same goal; deliver the best possible experience to the widest possible audience whether your users are viewing your content on a smartphone, a desktop or even hearing it on a screen-reader. Those strategies emphasizes accessibility, semantic HTML markup, external stylesheet and scripting technologies.

Graceful degradation is the older of those concepts. We can also find this term in other areas such as precision mechanics or electrical systems.
It is a technique by which it applies primarily to create features for the newest browsers, those capable of supporting almost all of the proposed solution. Then, graceful degradation, consist in considering older browsers by specifying least behaviors which are sensoriality completely different but which will be supported by them.

One example is to use the tags *<noscript />* to specify a different behavior if the browser that runs the page doesn't allow the use of JavaScript.

[3] Source : http://httparchive.org/trends.php#bytesTotal&reqTotal
[4] Source : http://httparchive.org/trends.php?s=All&minlabel=Nov+15+2013&maxlabel=Oct+15+2016#bytesTotal&reqTotal

```html
<script type="text/javascript" src="/menu.js"></script>
<noscript>
  <ul id="menu">
    <li><a href="/">Home</a></li>
    <li><a href="/products/">Produits</a></li>
    <li><a href="/services/">Services</a></li>
  </ul>
</noscript>
```

Another one is the alt attribute for image element. It has becomes a habit recommended by the W3C but it is primarily a means of allowing older browsers to display a default text if the image isn’t readable as well as an accessibility tool to give to the peoples with impairments, to understand the displayed content using screen readers, for instance.

Progressive enhancement consist in the first place to create understandable and interpretable files for all browsers. Then add visual enhancements and semantics for all browsers that can understand them. So it is about creating “basic” web pages which will be decorated by new items after.

For example, you can choose to create a gradient using a repeating image on an axis for a browser like Internet Explorer 7 and then decide to use the linear-gradient function to get more precise gradient which will be generated without image.

To sum up, progressive enhancement allows to have a user experience designed so that any browser and device will be able to offer the same user experience for everyone.

A Progressive Web App is a website or a web application which is developed prioritizing mobile experience before everything else.
When we are surfing on a PWA, everything is designed in order to have the best experience possible no matter what platform is used (browser or operating system).

In the case where the platform used would allow it, PWAs use the latest web technologies to deliver a superior experience, independent of connectivity, immersive identical to native application, installable after several uses that re-engage users.

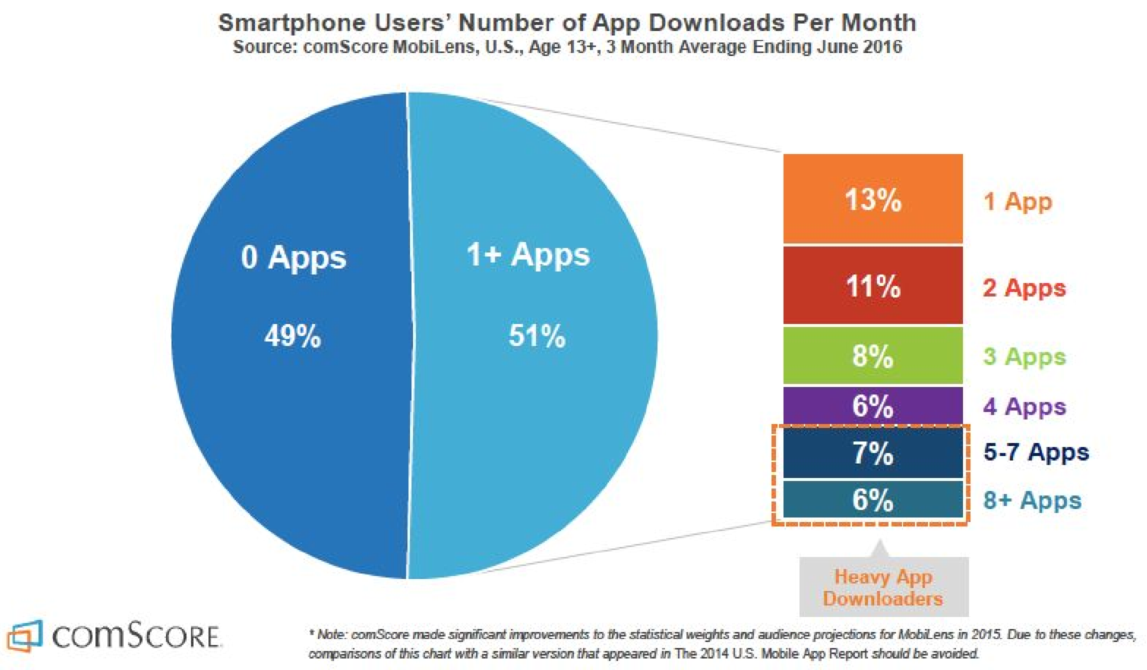
Following all those evolutions and new capabilities, **how Progressive Web Apps can revolutionize application development ?**

First, we will do a state of the art to draw the the distinguishes, strengths and weaknesses between Web applications and native applications. Then we will see the innovations that both platforms engaged to face those weaknesses. In a third part we will see the bad cost of those innovations and then try to bring our own solution to solve this problem and discuss on what is still to do in the Web, before we conclude.

I. State of the art

A. Native application constraints

1. High friction of distribution



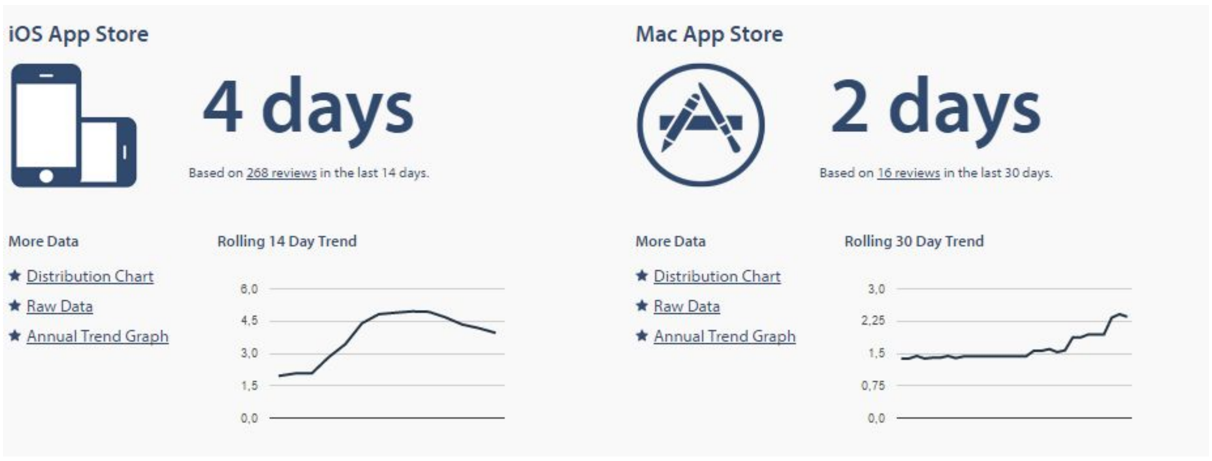
As you can see on the image above, vast majority of smartphone users fail to download any new apps each month. Even though people are using the apps they have more than ever, only half of U.S. smartphone users choose to give new app a try. Stores are growing larger and larger every day that's why it became really hard to connect the right user to the right app.

To download a native application, you have to go to the app store, search for the app, click install, then wait for the download to finish and finally open the app. *Each of these steps loses 20% of the potential users*[5].

[5] Source : http://blog.gaborcselle.com/2012/10/every-step-costs-you-20-of-users.html

2. App Store approval

Mobile applications developed for iOS are subject to approval by Apple for basic reliability testing and other analysis. If the application doesn't fit Apple's requirements, the author can still try to submit it after some changes.



Apple application approval time

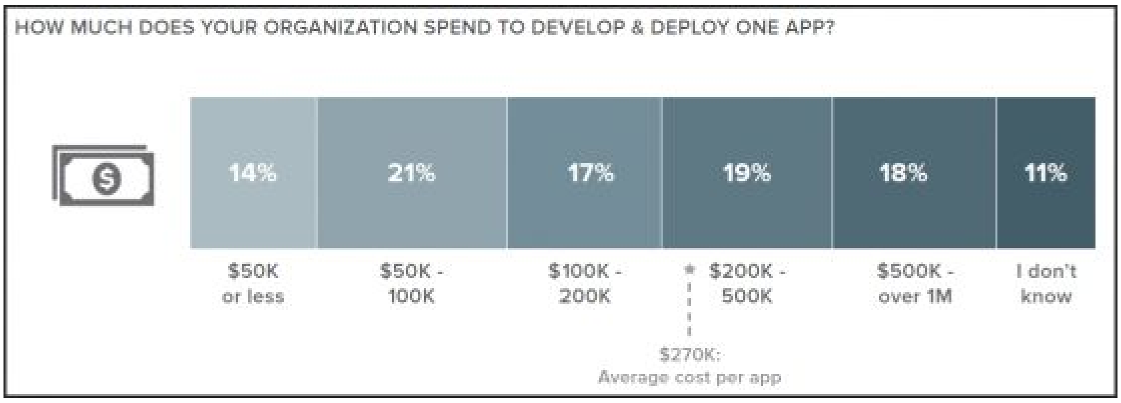
This process can be long and laborious since there is a lot of steps to get through and need not always result in success.

3. Cost

Developing a native application imply having to build multiple versions of the app to support a lot of mobile platforms and their versions in a highly fragmented market.

To sum it up, going native means significant development costs and time to market for a software company as native apps are not from an independent platform. For instance, an app developed for iOS won't work on Android or on Windows phones.

Furthermore, recurrent upgrades in the platforms induce in significant overheads for applications developers to test and address compatibility issues.
All these facts lead to a high development price.



[6]

Average cost per app

4. Limited access

Native application are limited in their access by some policies and geographic restrictions.
It is nearly impossible to fulfill all the conditions necessary for its application to be available all over the world.
This constraint can only be bypassed if the user uses a VPN or passes for an app store, which limits the friction of the application.

5. Data consumption

In some continents, like Africa, the vast majority of people use their mobile phone to surf on Internet. But poor connectivity, the prevalence of low end devices and several other obstacles can block the growth of an application.

On these countries, the cost of Internet is still high while average salary is still very low. Current and potential customers are very data-sensitive and many of them will hesitate before using data to download native app.

[6] Source : http://www.formotus.com/14018/blog-mobility/figuring-the-costs-of-custom-mobile-business-app-development#prettyPhoto

B. Front-web renewal

1. HTML5

HTML5 is the latest evolution of standards that define HTML. It is a markup language used for presenting and structuring content on the web. HTML5 was published in October 2014 by the W3C (World Wide Web Consortium) to improve the language with support for the latest multimedia while keeping it both easily and readable by humans and consistently understood by computers and devices.
The term HTML5 regroups two different concepts.

First, this is a new version of HTML language with new elements, attributes and behaviors. But also a broader set of technologies that enable more varied and powerful websites and applications. This set is sometimes called HTML5 & Cie and often juste abbreviated as HTML5.

Secondly, it has been designed to be used by all developers of Open Web.

There are many improvements in this new HTML version :

- Semantics: allows to accurately describe content.
- Connectivity: allows to communicate with the server in a new and innovative way.
- Offline and storage: allows web pages to store data on the client and run more efficiently in offline mode .
- 2D / 3D rendering and effects: allows more diverse presentation options
- Performance and integration: provides much greater power and better use of computer equipment.
- Device access: enables varied use of input and output devices.
- Style: allows authors to write more sophisticated themes.

2. ECMAScript 2015

Historically, the JavaScript language has been completely sidelined for a long time in the web page designing are and has been used primarily to achieve various animations.
Today things are totally different, henceforth we speak of web application fully controlled via JavaScript.
Unfortunately, the language was not designed to create complex applications that’s why it results with very intricate syntaxes but this changed with ECMAScript 2015 arrival.

ECMAScript 2015 was thought to create maintainable web applications, while maintaining compatibility with existing code. The idea was to add new features to the language.

Thus, the standard library enriched with new methods, but the most important is that the language adopted new syntaxes such as classes or modules allowing us to have a readable and structured code.

All these innovations will help to quickly and easily create web applications. Adding classes with an easy heritage system is a real plus and will result in a better code structuration.

3. Polymer

Before getting into Polymer’s definition, it is important to define some important terms:

1. Custom Elements

Custom Elements allow developers to create their own featured DOM elements. Despite the fact that authors could use non-standard elements in their documents for a very long time, with the possibility to add specific behaviour after the fact by scripting or similar, such elements have historically been improper and not very functional. By defining a custom element, the parser is informed on how elements of that class should respond to changes and how to build an element in a proper way.

The Customs Elements are part of the broad effort that has been made to "rationalise the platform", by defining existing platform features (like the elements of HTML) in terms of lower-level author-exposed extensibility points (like custom element definition). Despite the fact that today there are a lot of limitations on the abilities of custom elements, either semantically or functionally, we hope this gap will shrink over time.

2. HTML Templates

The HTML <template> element is a mechanism used to store client-side content and should not be displayed when the page loads but can be instantiated later using JavaScript.

This element is a fragment of content set aside for later use in the document. When the engine processes the content of the <template> element when loading the page, it only checks the validity of the content, which is not displayed.

The basic idea behind the template element is that it can be used as many times as we want. You can use the template as it is, or you could add to it, for instance, data from a database. Thus, template element can be used to insert rows into a table, each of which is built from the template, or a select list with new options perhaps based on user input.

Until the template element is used, it is not part of the document. Therefore, you can't reference it through the DOM, for example, using document.getElementById or similar code. It is also important to know that none of <template> contents are downloaded until it is used. That's why that if the <template> contains for instance a video, it will be downloaded when the template is duplicated and introduced into the document.

3. Shadow DOM

The Shadow DOM allows to encapsulate JavaScript, CSS, and to realize templates within a Web Component. The Shadow DOM aims to separate the DOM from the main document. The Shadow DOM can be used alone, outside the Web component.

Why search to separate the code from the rest of the page? The main reason are large size sites, for example when CSS is not organized, styles can overflow on parts of the site when it was not desired, and vice versa. When a site or application is grown, this constraint must be avoided.

4. HTML imports

A HTML import is a way to import and reuse HTML in HTML.
The imports are done via the same tag that we use to include CSS style, the <link> tag. More precisely in this way :

```
<head>
    <link rel="import" href="path/to/element.html">
</head>
```

It is possible to import several files, which can themselves import others (which may already be in the list of imported files), but whatever the import dependency tree, each imported file will be only one time. This will greatly increase our performance.
To manipulate the imported HTML, we have to access the contents of an imported file, one must still go through javascript. First we need to access our link element. Then, the DOM of the imported file is accessed via the content attribute of the link.

5. Material Design

Polymer makes it possible, for example, to apply the rules of Material Design in a web application.
Material Design is a set of rules that apply to the graphical interface of software and applications. It is notably used in the version 5.0 of the Android operating system.
This new design language is, according to Google, based on paper and ink.

"Unlike true paper, the digital material can stretch and change intelligently. The contextual material has a physical surface and edges. Overlays and shadows give information about what you can touch. " Matias Duarte, Designer and interface designer Google.

This concept created by Google aims to guarantee the ease of use of Android applications to make them simpler, more refined and more intuitive (user friendly).
Material Design is also Google's answer to the problems posed by contemporary design: an efficient design on all supports, a self-adaptive, interactive and intuitive web design for all users. The adoption of material design aims to unite with a unique and homogeneous design all the services and materials of Google.

The user experience has become today a necessary axis of innovation for brands already present on the Web in order to stick to the current trends in web design.

Polymer technology makes it possible to think development differently. Each web component is independent and is therefore easily reusable throughout the development of an application. This new technology avoids repetition and saves time.

4. NodeJs

JavaScript is mostly used for client-side programming purposes. As a cross-platform runtime environment, Node.js enables developers to run JavaScript on the web servers.

Node.js is quite different from other JavaScript frameworks frequently used. It interprets the JavaScript code thanks to JavaScript V8 Google engine.

Moreover, most basic modules of the open source runtime environment are written in JavaScript. That's why web developers can use Node.js to use JavaScript for developing frontend and backend applications.

Node.js allows developers to use a single programming language as we said earlier. JavaScript is now supported by both web browser and server that is why it becomes easier for companies to deploy their applications.

It is also highly extensible. Node.js already supports several commonly used web development tools but it is greatly extensible. Since the project is open source, it is easier for programmers to customize and extend Node.js according to their needs. They can also use JSON to make exchange between the web server and the client easier.

Node.js facilitates real-time application development such as games and chats since the runtime environment manages several concurrent connections effectively.

5. HTTPS

Hypertext Transfer Protocol Secure is an Internet communication protocol which protect the privacy and integrity of datas during the transfer of information between user device and the website.
Users expect to have a confidential and secure online experience when they visit a website.
Data sent by HTTPS protocol are secured via TLS (Transport Layer Security) protocol which provides three key levels of protection.

- Encryption: encodes the data exchanged to protect them from unauthorized interception. This means that when a user surfs a website, no one can "listen" to his conversations, follow his activities on various pages or steal his information.
- Data integrity: Information can not be changed or corrupted during its transfer, either deliberately or otherwise, without being detected.
- Authentication: proves that Internet users communicate with the right website. This method protects against the attacks of the interceptors and creates a climate of confidence for the Internet user which translates into other spin-offs for your activity.

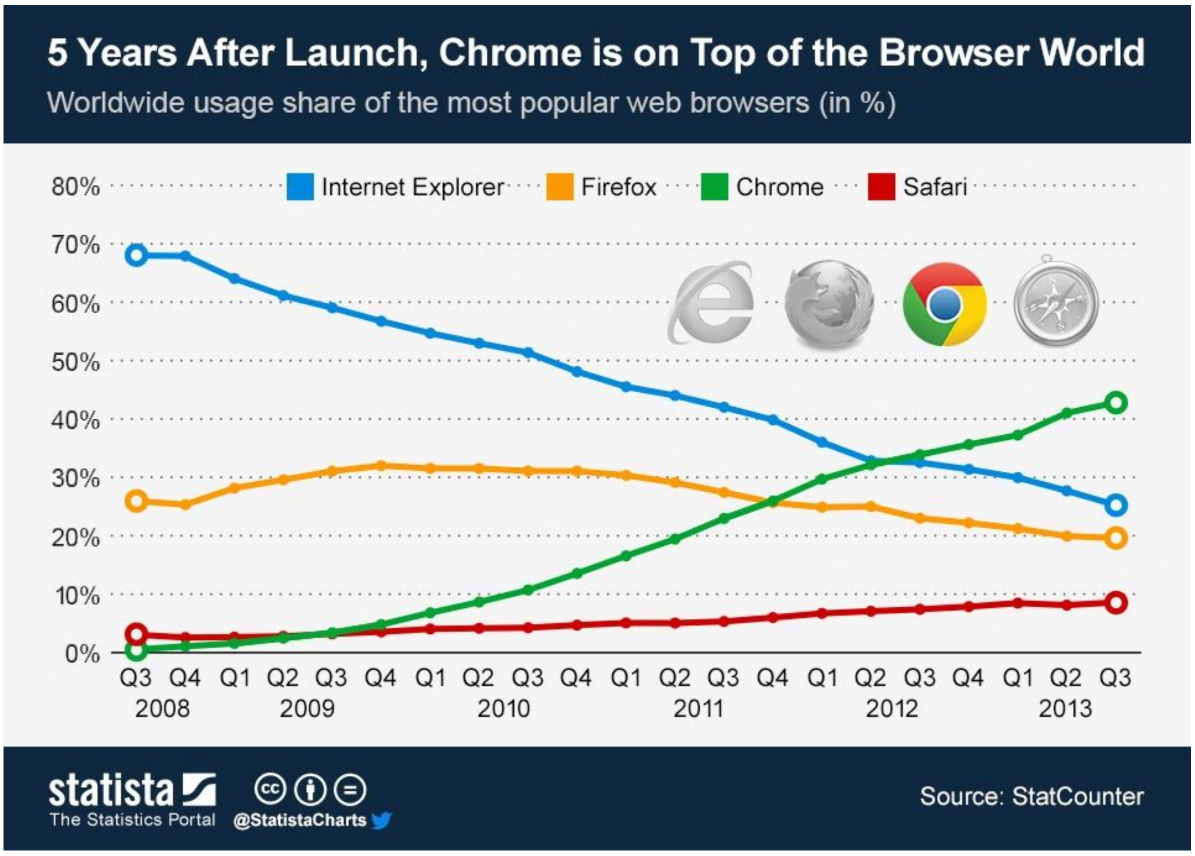
Nowadays, HTTPS is really important because some browser features are not available with HTTP such as Geolocation, HTTP/2, Push notifications and others will be removed very soon. It is also a requirements to build a Progressive Web App as we will see later.

C. Chrome as a platform

Chrome is the web browser developed by Google. This browser has been on the market since 2008 and runs on different platforms such as PC, tablets, or smartphone, and under different OS Windows, Mac, Linux, Android.

Just like Firefox it is part of an open-source project, in this case The Chromium Projects. Being part of the google eco-system, it has a web store allowing users to access applications like Google Drive for storage in the cloud, themes and many other tools. With Microsoft Internet Explorer, Mozilla Firefox, Google Chrome is one of the three most widely used browsers in the world.

Browser are the underlying power of the Web, and as Web developer, our platform.



1. Chromium project

Chromium is an open-source web browser that is used by Google Chrome and other proprietary Web browsers.

The Chromium Project takes its name from the element chromium, the metal from which chrome plating is made. Google's intention, as expressed in the developer documentation, was that Chromium would be the name of the open-source project and that the final product name would be Chrome; however, other developers have taken the Chromium code and released versions under the Chromium name. These are listed under community packages.

One of the major goals of the project is for Chromium to be a tabbed window manager, or shell for the web, as opposed to it being a traditional browser application. The application is designed to have a minimalist user interface. The developers state that it "should feel lightweight (cognitively and physically) and fast.

Chromium is a web browser development project of the same name. Some users prefer Chromium to Google Chrome because they deem Chromium's contributions to Chromium (especially tracking features) unnecessary or detrimental.

To generate Google Chrome, Google uses Chromium's source code and adds the following functionality:

- Default integration of Flash Player8;
- Integration of an internal PDF reader with the browser9;
- Addition of a print and print preview system;
- Change of logo and addition of the mention of Google;
- Added an automatic update system called GoogleUpdate;
- Added an option for users to send their usage statistics and error reports to Google;
- Addition, in some cases10, of tracking features called RLZ tracking;
- Added support for AAC and MP3 proprietary codecs (Chromium only supports free Vorbis, Theora and WebM codecs).

2. V8 engine

JavaScript is ubiquitous on the Web. The developers use it to dynamize the content of web pages, and thus significantly improve the user experience.

Each browser implements its own home-made solution for the interpretation and execution of JavaScript scripts. Safari uses JavaScriptCore, Microsoft uses the Chakra engine, Mozilla develops Rhino and SpiderMonkey, Google for its part set up V8.

Google developers have opted for a compilation on the fly (JIT Compiler) for V8, which allows to convert language scripts to native machine code so that they can run faster.

This conversion has a price: it increases the response time of the pages. This is why the JIT compilation of V8 is done in two steps. The first runs just before a script runs. It is fast, but does not include a phase of optimization of the native machine code.

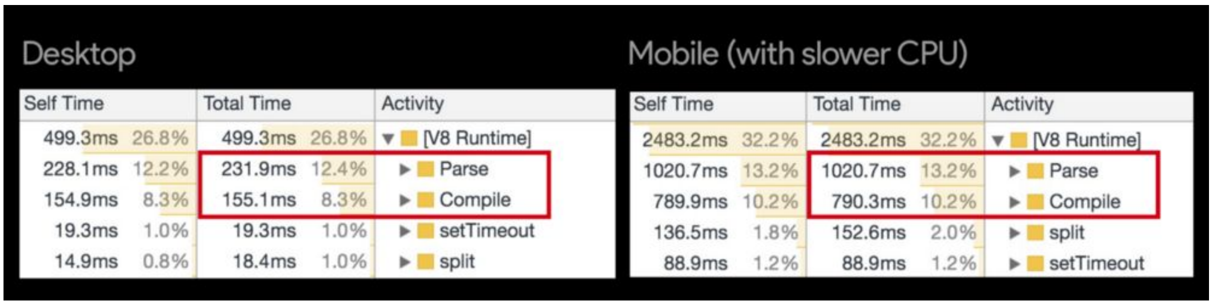
On the other hand, the second step of compilation takes place more slowly on scripts that must be executed more than once. This is the optimization phase.

If for most scenarios this approach proves to be ample, in some situations it is ineffective. This is particularly the case for complex applications like games, or the second phase of compilation causes an irrefutable increase in the response time of the page.

Google developers have been working on the issue, and have come to the conclusion that using competing compilation would solve the problem.

In practice, the two on-the-fly compilation phases occur at the same time, except that the optimization step is performed in a competitor thread in the background.

During web apps start-up, parsing, compiling and executing scripts can take a long time. This delay can lead to a bad user experience. For instance, users could see a button but won't be able to click on it for several seconds.



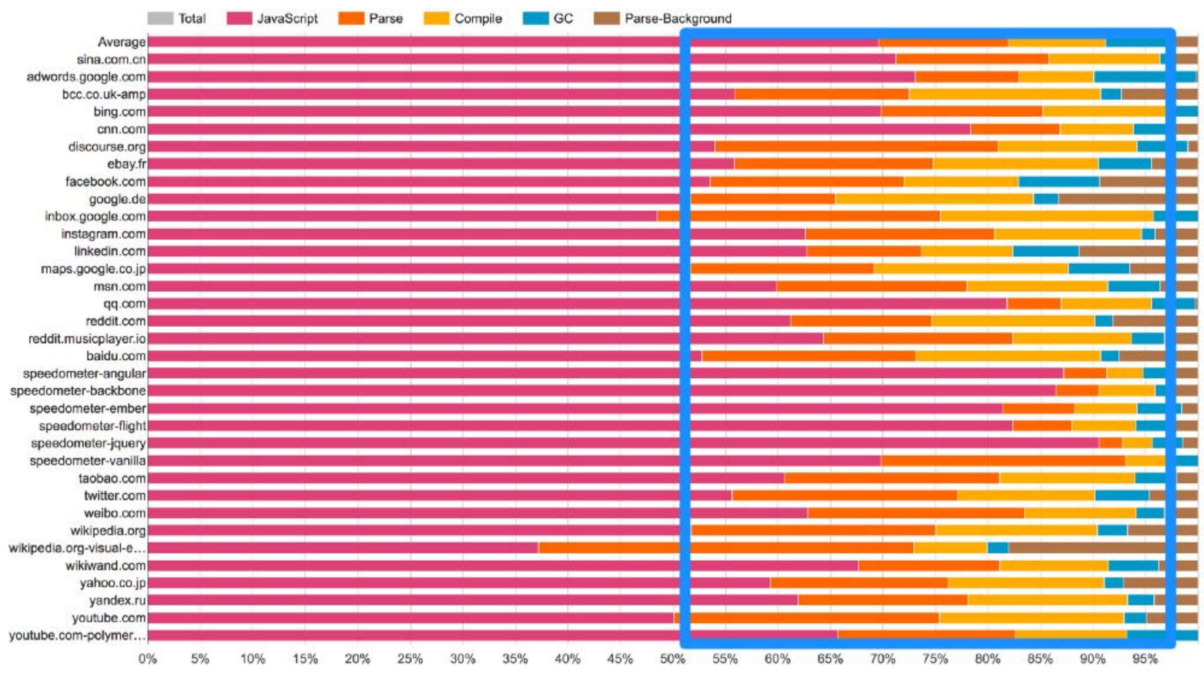
Desktop

Self Time		Total Time		Activity
499.3ms	26.8%	499.3ms	26.8%	▼ [V8 Runtime]
228.1ms	12.2%	231.9ms	12.4%	▶ Parse
154.9ms	8.3%	155.1ms	8.3%	▶ Compile
19.3ms	1.0%	19.3ms	1.0%	▶ setTimeout
14.9ms	0.8%	18.4ms	1.0%	▶ split

Mobile (with slower CPU)

Self Time		Total Time		Activity
2483.2ms	32.2%	2483.2ms	32.2%	▼ [V8 Runtime]
1020.7ms	13.2%	1020.7ms	13.2%	▶ Parse
789.9ms	10.2%	790.3ms	10.2%	▶ Compile
136.5ms	1.8%	152.6ms	2.0%	▶ split
88.9ms	1.2%	88.9ms	1.2%	▶ setTimeout

We can see in these tables that parsing and compiling scripts (popular websites) take much more time on mobile devices since they have a slower CPU.



[7]

In the above chart, pink area represents V8 execution time. We can see that parsing and compiling is almost taking half the time of the start up execution script.
Using modern building techniques like code-splitting, tree-shaking and service worker caching can really make an important difference.

[7] https://medium.com/dev-channel/javascript-start-up-performance-69200f43b201#.azvm1owa4

3. Web worker

JavaScript is a language that is based on a single thread. Even an AJAX call, although asynchronous, runs in the same thread as the rest of the application. For a long time, JavaScript has been criticized for this gap. This is no longer the case since 2015. This slide of a conference by Brendan Eich, the creator of JavaScript, dates from 2012 and illustrates this.

First of all, let's demystify what Web Workers are not.

- Web Workers are not Service Workers: the opposite is true. The purpose of a Service Worker is to act as a proxy between the browser and the outside, allowing, among other things, to handle a cache and have a pleasant offline experience. A Service Worker works only in HTTPS for security reasons, but Web Workers do not.
- Web Workers are not meant to manipulate the DOM. So this is not the place to edit your page or import jQuery.
- Web Workers are not another way to make an AJAX call.
- A Web Worker will not have access to the alert or prompt functions but you can quite use console.log.

Apart from that, you have access to the vast majority of the existing API in the browser.

A Web Worker is a way to call a JavaScript script that will run in a different thread than the script that creates it. It is possible for both scripts to communicate together by sending and receiving messages. A Web Worker is however not lightweight and is not to be misused.
Creating is an expensive process, especially in memory. It can however greatly relieve the display and the update of a web page if you have to perform complex calculations or regular operations that can not change anything on your page. Typically, if you want to do regular checks on an API and compare your cached data to your displayed data to find out what you're going to change, it may be a good idea to use a Web Worker.

4. Cache APIs

Loss of connection is a major problem that web users have faced for years. The best web application in the world would produce a very bad effect if you could not download it. There have been many attempts to implement technologies that could solve this problem, and some issues have been resolved. But the main problem remains the lack of a global control mechanism for caching resources and customizing network queries.

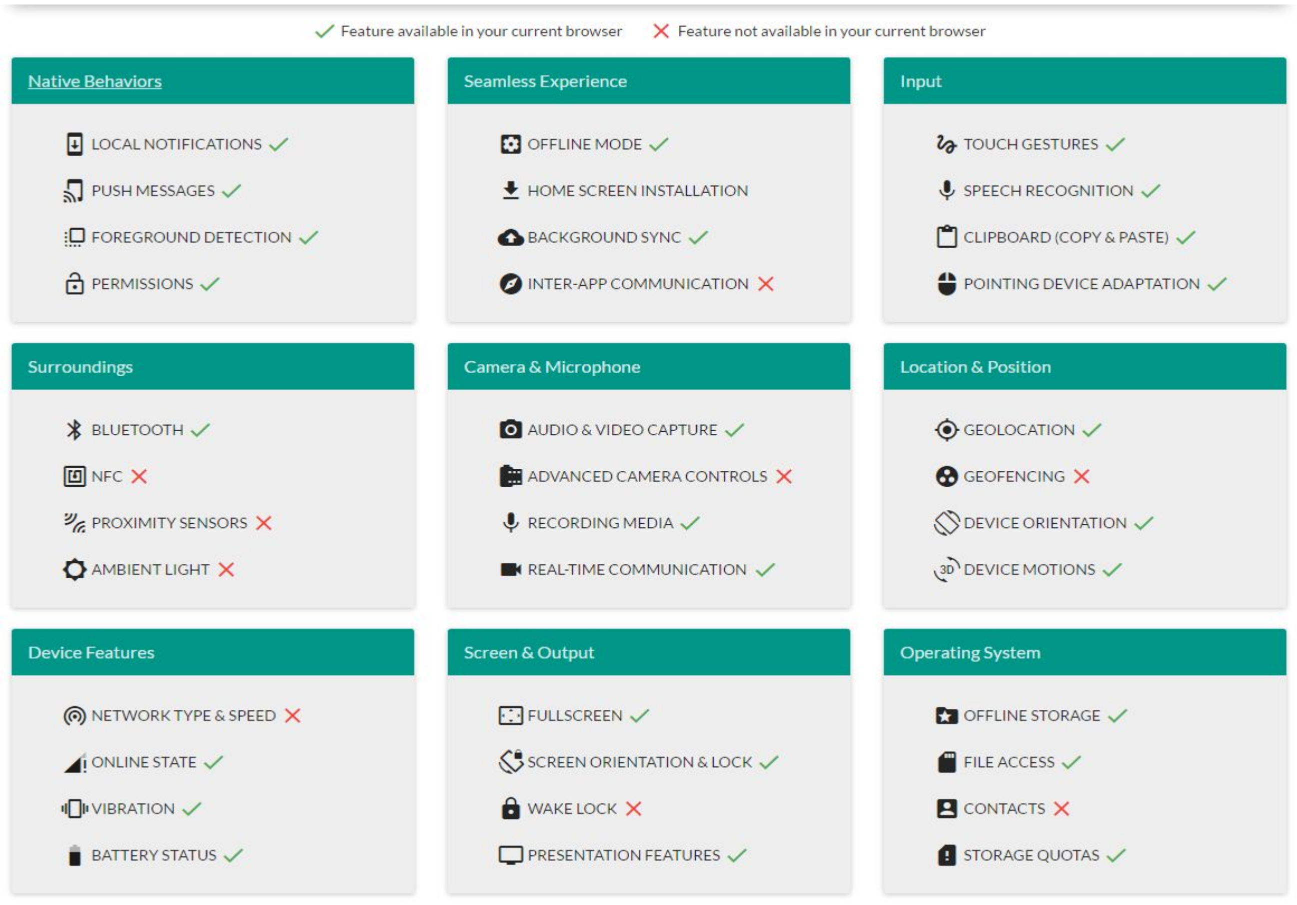
The previous attempt - AppCache - seemed to be a good idea because it made it possible to specify the resources to cache in a very simple way. However, she made many assumptions about how to set it up and failed relentlessly if your application did not exactly comply with these assumptions.

Service workers should ultimately solve these problems. The syntax of the Worker service is more complex than that of the AppCache, but you can use JavaScript to control the behavior caused by your local cache with much finer granularity, enabling you to address this problem, other. With the help of a Service Worker, you can easily configure an application to first use cached resources, providing a first default experience even in disconnected mode, before retrieving more data from the network Generally known as Offline First). This is already available with native applications, which is one of the main reasons for the preference given to these applications rather than to web applications.

To conclude, we can say that App cache was a great Idea and brought a revolution on the way developers could cache their content and deliver it with no internet connectivity, but the implementation was no broadly accepted by all browser vendor and make it impossible for developers to really use it.
Service workers bring a more unified APIs and a more precise control over cached content.

5. Access to device features

The mobile web try to catch up for few years in terms of user experience compared to native applications. In front of the fact that a large part of the world uses mobile web to access the web and the problems of connectivity that we all know, it was important to respond to these problems.



✓ Feature available in your current browser ✗ Feature not available in your current browser

Native Behaviors	Seamless Experience	Input
LOCAL NOTIFICATIONS ✓	OFFLINE MODE ✓	TOUCH GESTURES ✓
PUSH MESSAGES ✓	HOME SCREEN INSTALLATION	SPEECH RECOGNITION ✓
FOREGROUND DETECTION ✓	BACKGROUND SYNC ✓	CLIPBOARD (COPY & PASTE) ✓
PERMISSIONS ✓	INTER-APP COMMUNICATION ✗	POINTING DEVICE ADAPTATION ✓

Surroundings	Camera & Microphone	Location & Position
BLUETOOTH ✓	AUDIO & VIDEO CAPTURE ✓	GEOLOCATION ✓
NFC ✗	ADVANCED CAMERA CONTROLS ✗	GEOFENCING ✗
PROXIMITY SENSORS ✗	RECORDING MEDIA ✓	DEVICE ORIENTATION ✓
AMBIENT LIGHT ✗	REAL-TIME COMMUNICATION ✓	DEVICE MOTIONS ✓

Device Features	Screen & Output	Operating System
NETWORK TYPE & SPEED ✗	FULLSCREEN ✓	OFFLINE STORAGE ✓
ONLINE STATE ✓	SCREEN ORIENTATION & LOCK ✓	FILE ACCESS ✓
VIBRATION ✓	WAKE LOCK ✗	CONTACTS ✗
BATTERY STATUS ✓	PRESENTATION FEATURES ✓	STORAGE QUOTAS ✓

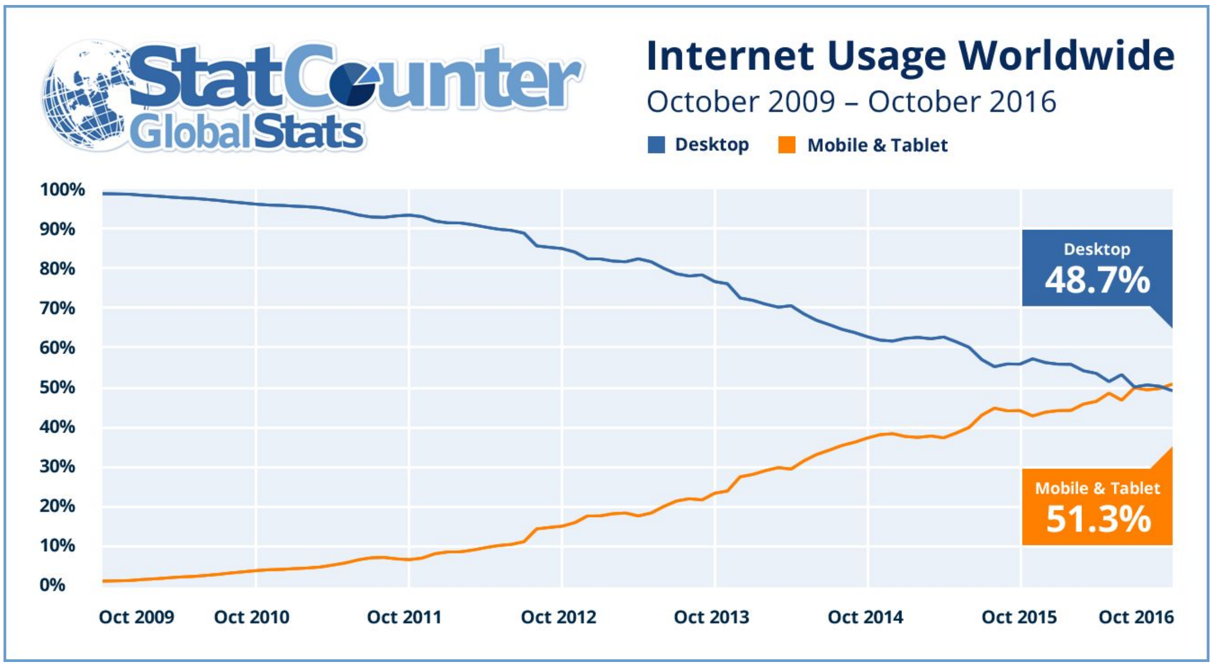
As you can see on the image above, the mobile web can access a lot of features of mobile devices and some of missing features are currently being implemented :

- Bluetooth via Web Bluethooth API (not supported by default but can be enabled in latest versions of Chrome, Opera and Chrome for Android)
- NFC (feature not on a current W3C standarts track but it is supported on some browsers)
- Ambient light sensor (supported by Edge and partially supported by Firefox)
- Proximity sensors (supported by Firefox)

8 https://stackoverflow.com/questions/35504194/what-features-do-progressive-web-apps-have-vs-native-apps-and-vice-versa-on-an

II. Innovation

1. Android Instant Apps



Today, as you can see on the chart above, people are more likely to surf the web with their mobile.
The disadvantage is that for some services, it is hard to offer a presentable experience on the mobile web. That's why some companies will go to any extreme, even harassing users with hideous interstitials for their applications to be installed on as many mobile devices as possible.

Google, however, unveiled a feature of Android that could change this during its I/O conference at Shoreline Amphitheater in Mountain View in 2016. It is called "Android Instant Apps" and as its names indicate, it allows to use apps instantly, without going through the whole installation process on the Play Store.

Instant Apps allows the user to download (in the background) only a part of a service application. For instance, if you come across a link of a service, it will download only the part of the application that matches the content you want to see. If you come across an e-commerce link on Google, Instant Apps will download the part of the online selling application that matches the product that caught your attention.

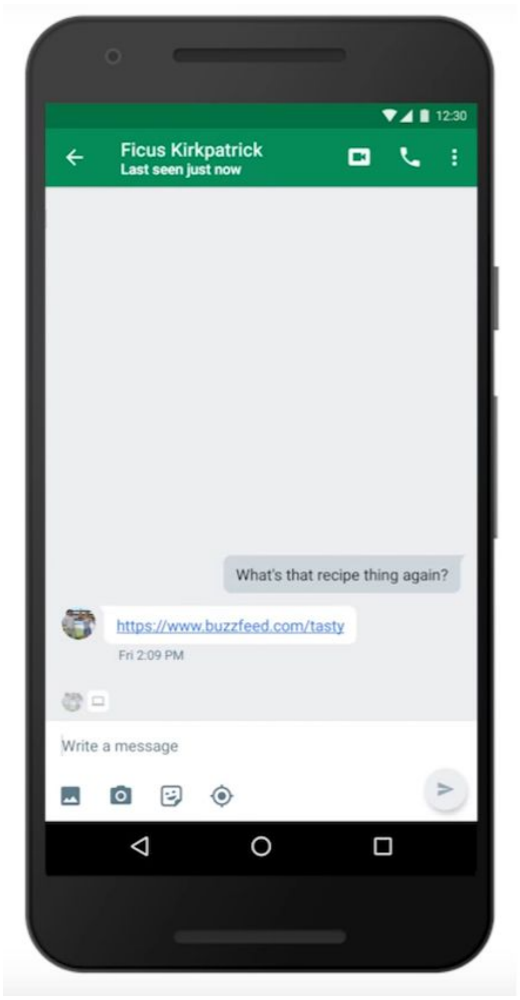
Then, it will even allow you to make a purchase, all without having to wait several seconds or even minutes to download the entire application.

Instead of rendering the store page in the web browser, Android will either load the service's app if it is installed already, or download a lite version of it to the device that is used only for that occasion but not installed on the device.
Downloads are limited to a maximum of 4 Megabytes for apps in this case, and users get the same experience that the application provides without installing it. In fact, the downloaded application is removed from the system afterwards again.

The expected result is that apps can launch instantly via Android deep links like web pages, whether already installed or not. And of course, if the user is satisfied, he will always be able to install the entire app later.

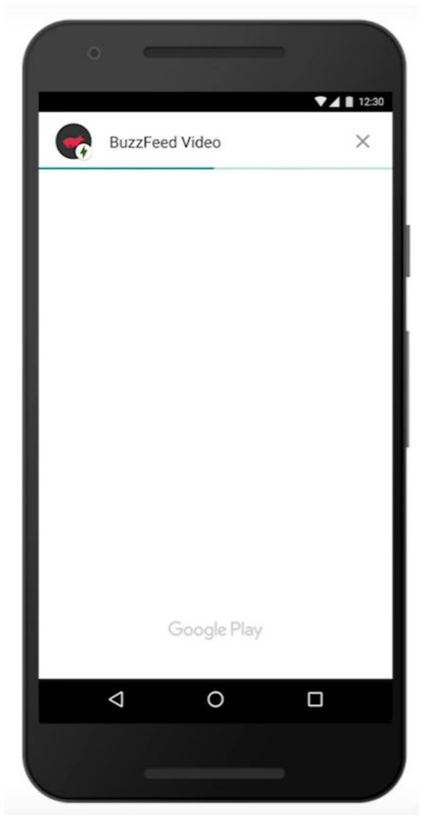
To better understand the function of Instant apps, let's take an example that we could see in the near future.

Imagine that we receive a deep bond sent by a friend as we can see in the picture below.



After clicking on the link, it redirects us to a download link. This download is almost instantaneous because the user does not download the complete application but only some of the code necessary for his needs, which is called an Activity.

An activity is the main component of an Android application. Activity is the job of the application and generally has a View at least, that is to say a graphic screen. So in a standard application, you could find an activity that lists contacts, an activity that adds a new contact, and an activity that displays the detail of a contact. The whole forms a coherent whole, but each activity could function autonomously.



Once the download is complete, the application tip starts and all this without installation. In this case, the user will be able to scroll the page as if it were in the application.

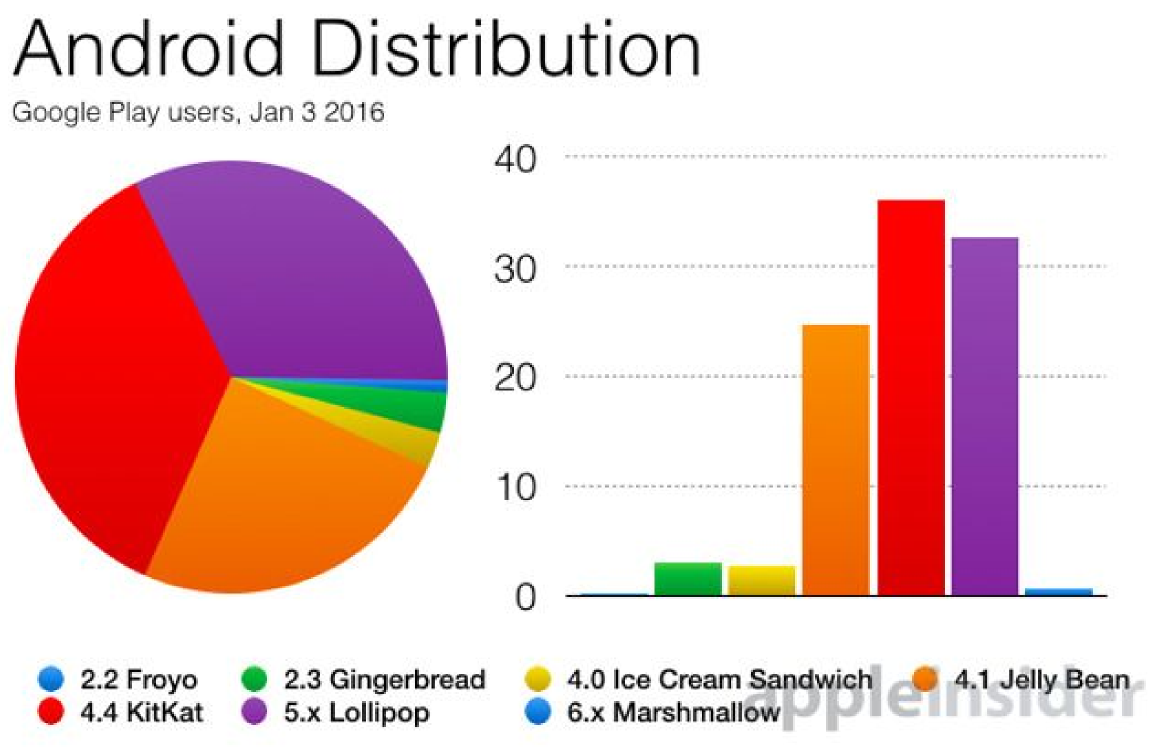
12:30
INSTALL
TASTY
Tasty
Food that'll make you close your eyes, lean back, and whisper "yessss."
Ice Cream Donut Holes

Devices that support Android Instant Apps

Google plans to build solutions for the future world that’s why it wasn’t surprising to learn that they started to work on Android Instant Apps with the Android 4.1 release. When Android 2.2 arrived, it was clear that Android was intented to be the most popular operating system on earth.

Google had to face the issue of the support of its operating system. On Android 2.2 release, it was obvious that Android device constructors often built smartphones to meet to the minimal specifications to power a device. It created a culture that obliged users to upgrade their OS frequently.

The major issue is that users are upgrading their devices approximately every 12 months. According to Google’s data, the most used Android version is Android 4.4 (Kit Kat).



This is why Android Instant Apps will run on any devices powered by Android 4.1 or higher.

Update existing apps

Android Instant Apps works with Google Play Services, current Android code and Android APIs. It will take approximately 1 day for an experimented developer to update a medium-size project to make it support Android Instant Apps features.
However, Android Instant Apps only works for apps built using native code (Java or C++). Apps build with tools and services for developing hybrid mobile apps like Cordova, Phonegap won't support it.
It requires that you build Instant Apps with Android Studio.

Security changes

The main challenge for Android is the increased perception of inadequate security.
Android Instant Apps seems to perpetuate the issue with apps being installed ad-hoc (It basically means, more or less, something that was made up on the fly just to deal with a particular situation, as opposed to some systematic approach to solving problems).
Thankfully, Google works on Instant App security for several years. Google Play is essential to Instant App success that's why Google improved considerably Google Play tools to detect malicious code. Only apps published on Google Play App Store will support Instant Apps.

This is a big step for security. There are a lot of independent Android app stores and most of them don’t have enough security measures as Google. By using Instant Apps, users know that the source of the app is Google Play and that the app has a prescribed level of security.

With Android Instant Apps, Google provide an excellent occasion to all web and Android developers to create web applications that works like their Android app equivalents. Developers will no longer have to choose between the reach of the web and the rich functionalities of native Android apps.

2. Progressive Web Apps

In term of innovation, the Web platform have come far from juste consulting documents online on a desktop computer, as mentioned earlier PWAs are about great user experience on mobile which is where the Web doesn’t shine, because of low end devices and bad overall connectivity.

This is the reason why we, as Web developers, have to stand up and bring something new to the user, and with all the innovations going on in the Web platform, it today easier than ever to enhance the Web experience.

As a reminder, Progressive Web App is a website or a web application which is developed prioritizing mobile experience before everything else. Because the next billion users that will interact with the Web will be using mobile devices.
When we are surfing on a PWA, everything is designed in order to have the best experience possible no matter what platform is used (browser or operating system).

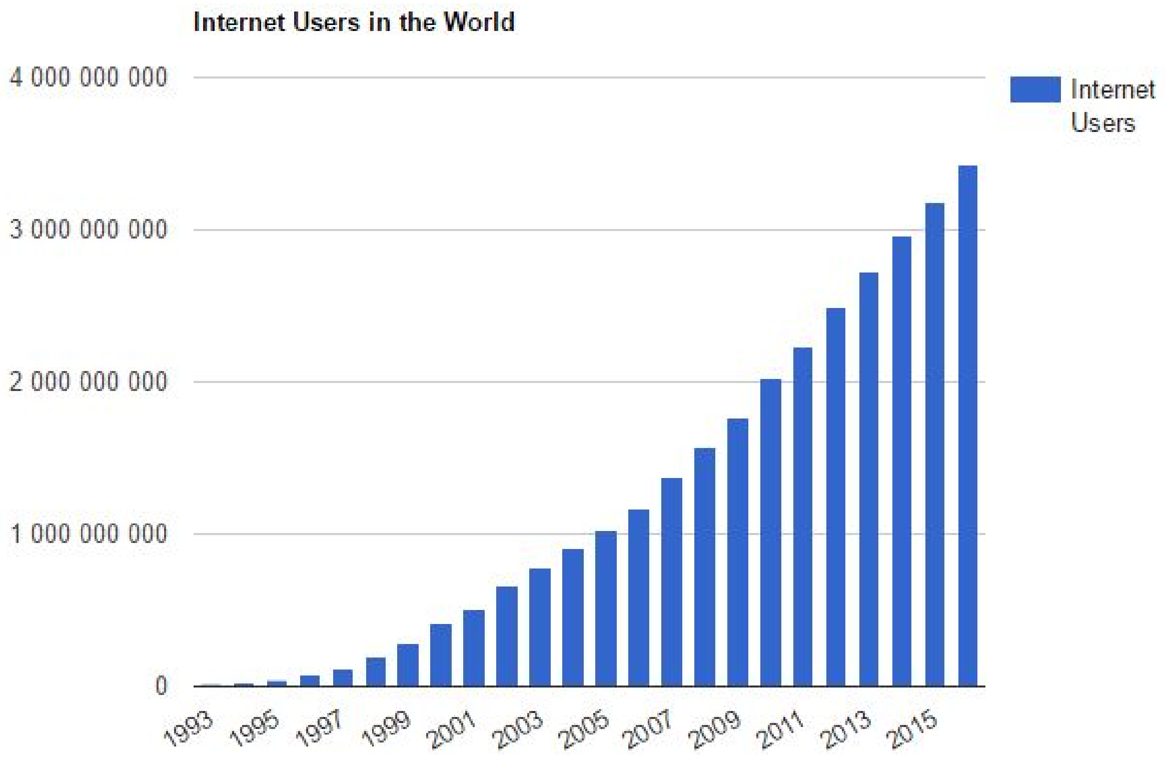
In the case where the platform used would allow it, PWAs use the latest web technologies to deliver a superior experience, independent of connectivity, immersive identical to native application, installable after several uses that re-engage users. PWAs are designed to bring offline first application to the Web platform.

1. The Web for the next billion users

As developers, we often think about who is going to use our application, and where are located our users. And sadly we have the mental image of our users as restricted as what we knew, often lead to a world as restricted as few developed countries.

The earth with only some developed countries.

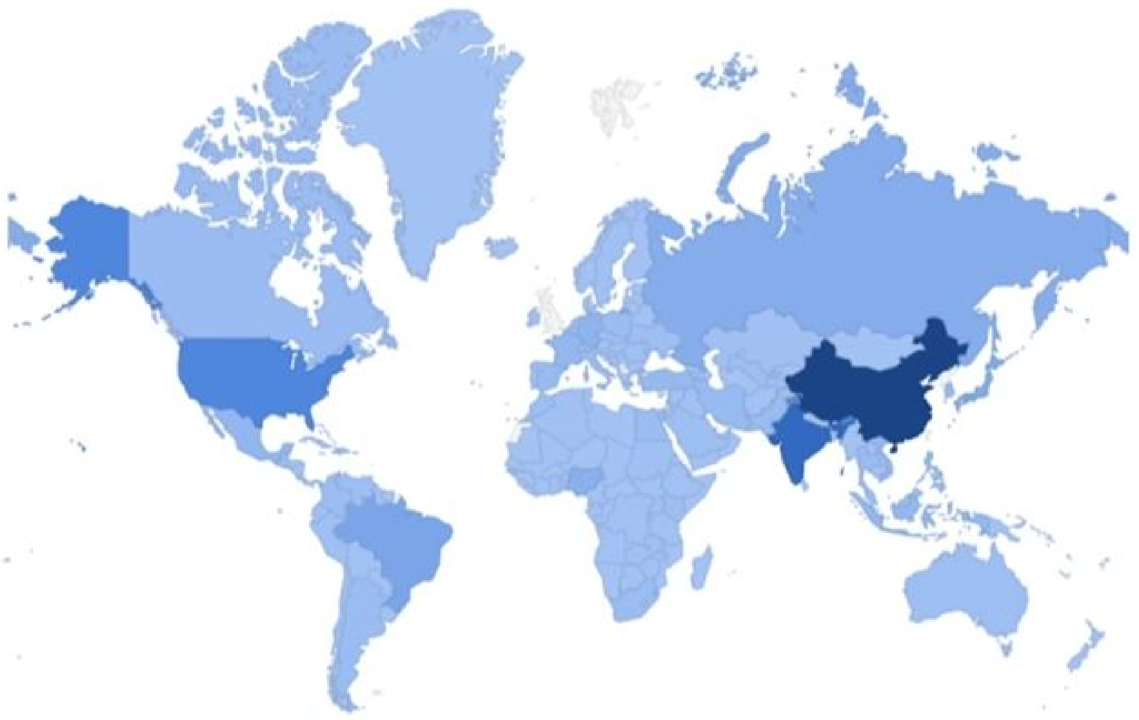
And this absolutely not represent all internet users in the world. As said earlier, we reach over tree billions internet users and this number will keep growing.



[9]

When thinking about, where users are located in the world, it's more spread than we often think.

[9] Source : http://www.internetlivestats.com/internet-users/



Internet usage concentration

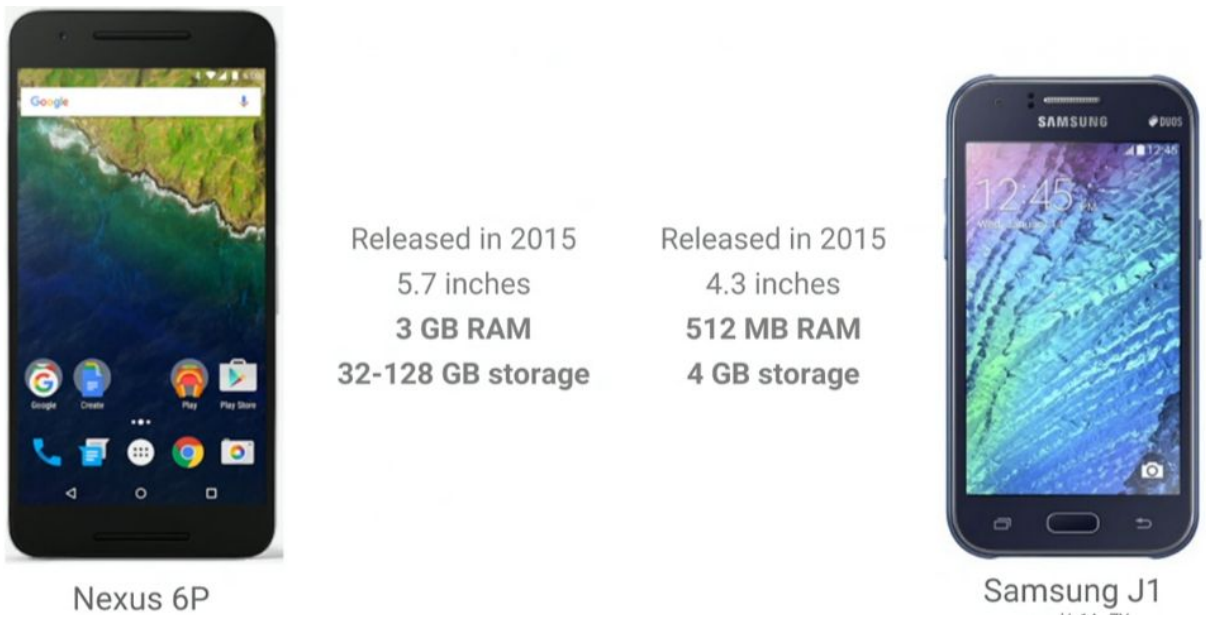
And we can see that there is a great concentration of users in Asia and other developing countries. This is where the next billions of users we have to target will be.
We know that good user experience also mean flawless interaction with the applications, and as Web developers, to do so we use Javascript. But something to keep in mind is that, on desktop machine, the Web browser can use all the power provided by the machine to run the javascript and then trigger all our animations and transition to make the application enjoyable, but as obvious as it seems, mobile devices are not as powerful as our desktops computers.

Earlier we said that the Web platform fails to adapt to its new environment. And this is true, if we think about how native platform adapt itself to the device the are targeting when transitioning to mobile devices.

Application developers don’t just take the same code they were running on desktop machines and through it on smartphones to run it, they adapted their codes and applications to fit in the new platform, even created new tools and programming languages, to make sure their applications can perform well on those platforms. No one takes a Java JAR file and directly run it on Android, even on Windows Phone, developers adapt their binaries and executables to run on less powerful device, not just launching a .exe file on the phone and expect it to run just as similar than on desktop.

[10] Source : http://www.internetlivestats.com/internet-users-by-country/

Mobile devices are so widely inequivalent in term of capacity and power. Even native developers have to build and think to specific device target to ensure that their application will run correctly, because, if in our developed countries, the majority of people are accessing our platforms through powerful devices, in other countries, it’s a totally different game.



Comparison between high end and low end smartphones

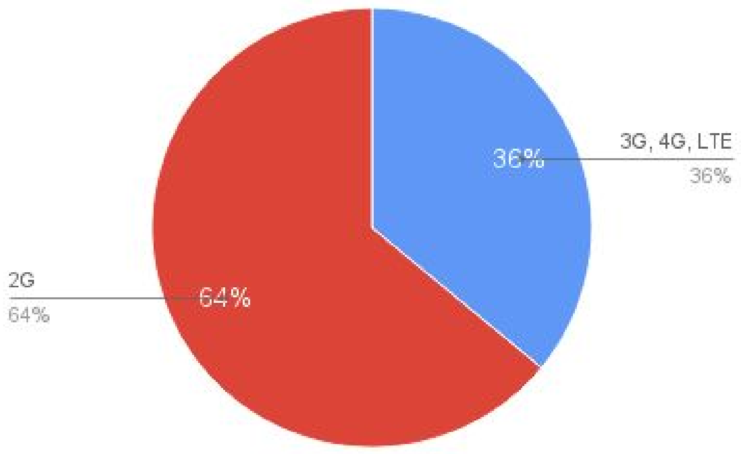
Average population in developing countries are using low end and slow devices like the Samsung J1. In comparison with the Nexus 6P which is alike smartphones we use in developed countries, this is a less powerful device, by all points, and yet, we expect it to run our code exactly the same way it does on powerful devices, and even worth we expect it to do so on almost no connectivity.

We have to really care about the next billion of people that will use internet in the future because, build a better Web for them ultimately will make us create a better Web for today's users as well.

2. Application Shell pattern thank to Service Worker and cache

Speaking of connectivity, one of the thing that make the most difference between Web apps and native apps is the ability to launch the application with no internet connection, the app will launch as normal but will likely struggle displaying contents if there is no cached data. That result of the way app are installed, app developers can save all necessary contents and not download them on app start. This way on low or no internet connection, the user is not forced to stare at a blank page loading contents but is immediately in an immersive user interface with visual elements that give him insights on what's going on. This minimal UI is the Application Shell.

Earlier we spoke about bad connectivity in developing countries, but bad internet connectivity is everywhere.



[11]

Average mobile connection speed

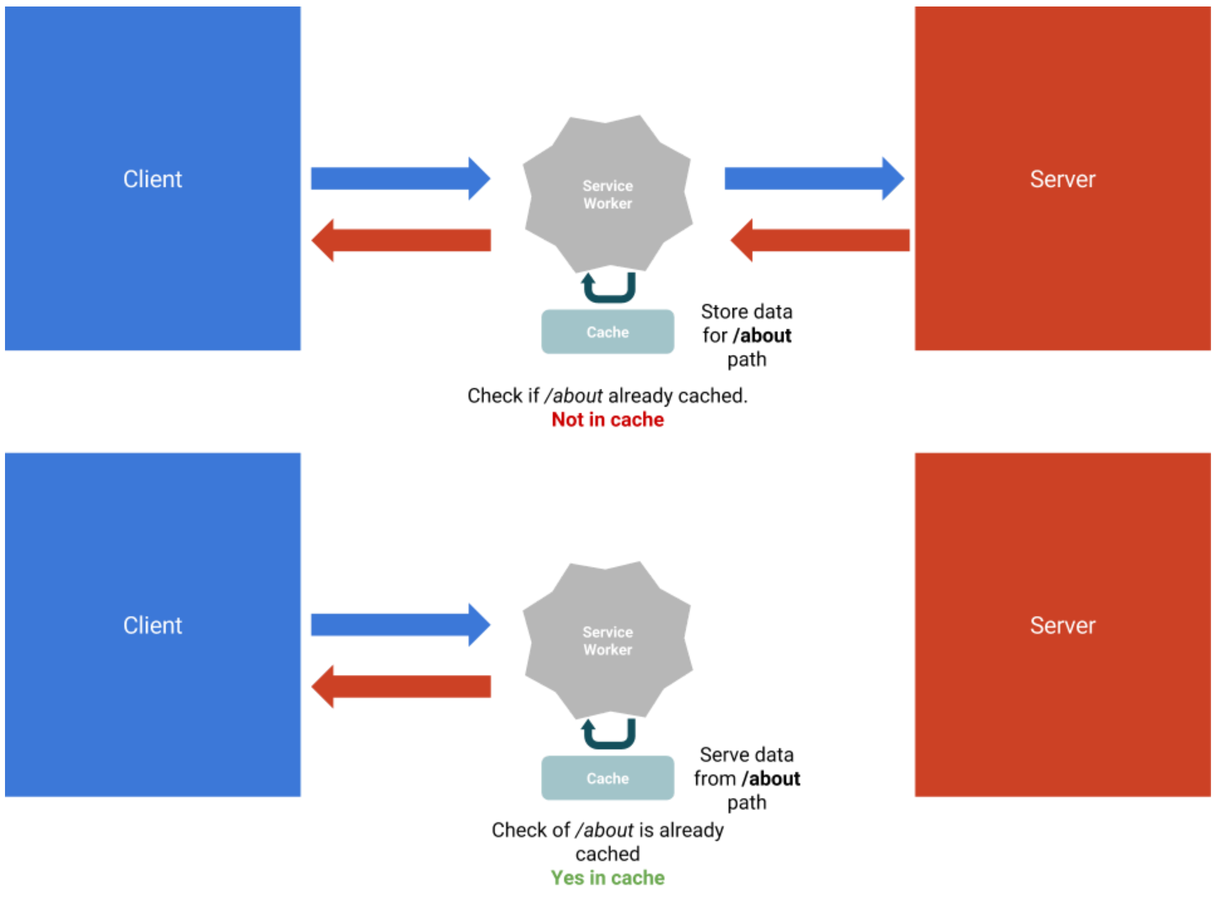
More than 60% of mobile connections globally are over 2G. With a connection that slow, pages load can takes several seconds or even minutes. This makes the Web absolutely uncompelling in regards of native apps.

To solve this issue, Web developers have take advantage of the recent improvements of the Web platform with APIs such as Web workers and find new ways to serve data with no connectivity. Those improvements led to the creation an unified API that gives new powers to Web developers. Formerly named Navigation Controllers, the Service Worker represent all the efforts make by browser vendors and developers to enhance the Web platform and compete with native apps.

As a reminder, services workers allow developers to create sites that work offline by intercepting network requests to provide a cached or pre programmed response.

[11] Source : http://www.strategyand.pwc.com/reports/connecting-the-world

The Service Worker act as network proxy between the client Web application and the server, allowing developers to handle the HTTP requests and decide or to not to let it reach the server to fetch new data.


Client
Service Worker
Server
Cache
Store data for /about path
Check if /about already cached.
Not in cache
Client
Service Worker
Server
Cache
Serve data from /about path
Check of /about is already cached
Yes in cache

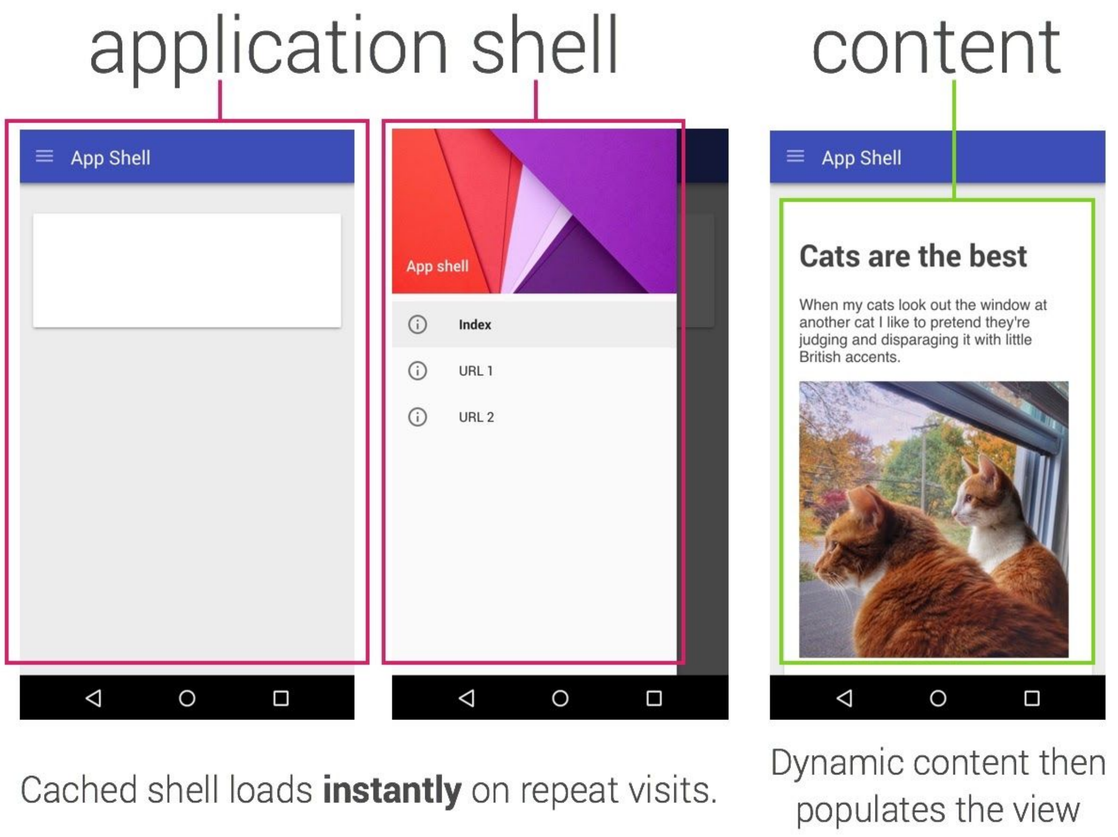
Service Worker and cache interface together

With those new possibilities, developers comes with a new design pattern that is the App Shell. As said earlier, where Web apps doesn't shine is by letting the user waiting on a blank page while loading the application. With Service Worker and caching APIs, it's now possible to make this white screen disappear. Of course, the user will have to download a first time the app shell, and then the Service Working will activate and cache it so the next time the user interact with the Web application, he only fetching new data while looking at the previously cached App Shell. To make that experience seamless, the App Shell have to load extremely fast even on bad network.

It's important to note that the Service Worker and manifest is only made possible by the secure connection provided by HTTPS, this way the system is sure that the content it will add in cache and add to the user home screen is secure.

3. Instant loading

By prioritising loading a minimal UI before anything else, while progressively fetch the rest of the data, Web developers can achieve Instant loading and significantly speed up the first paint of their application and free the users form the cursed white screen, and even better, with caching data thanks to the Service Worker, the application will still be available even offline.


application shell
content
App Shell
App shell
Index
URL 1
URL 2
App Shell
Cats are the best
When my cats look out the window at another cat I like to pretend they're judging and disparaging it with little British accents.
Cached shell loads **instantly** on repeat visits.
Dynamic content then populates the view

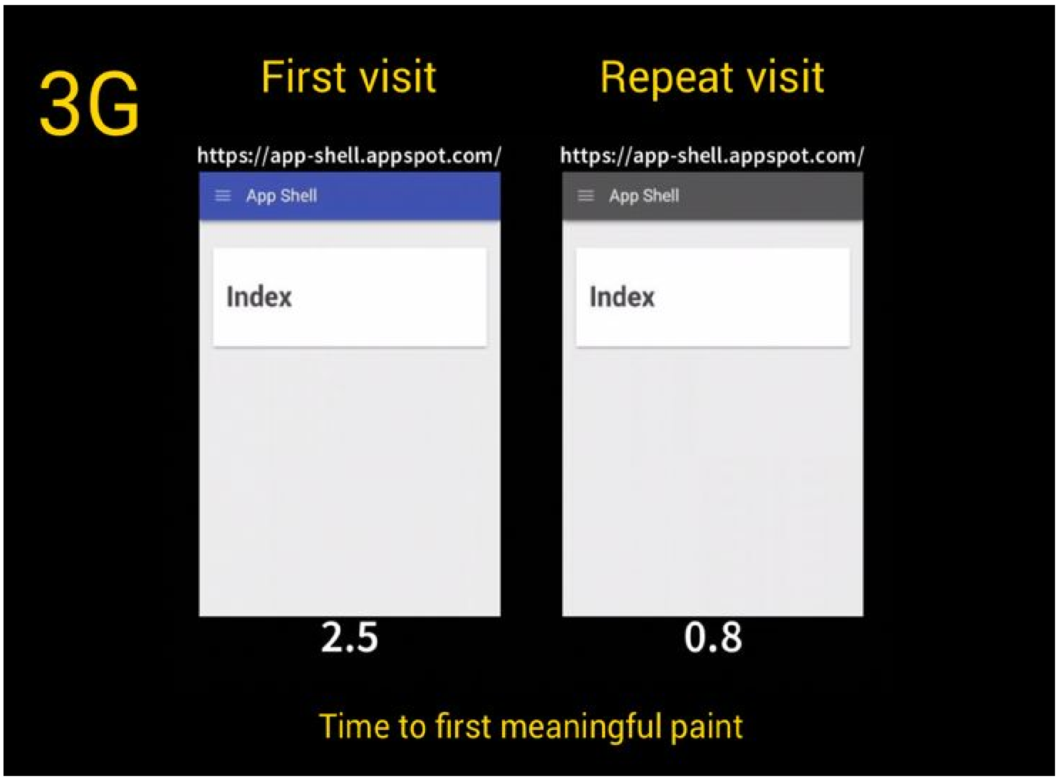
[12]

Application Shell pattern

This is an example of application shell, it does not require a lot of thing, It's basically a minimal UI with a navigation bar and a menu. To be sure that those elements load really fast, we use inline CSS styles, that way we are saving our app to make another HTTP call to fetch the necessary styles.

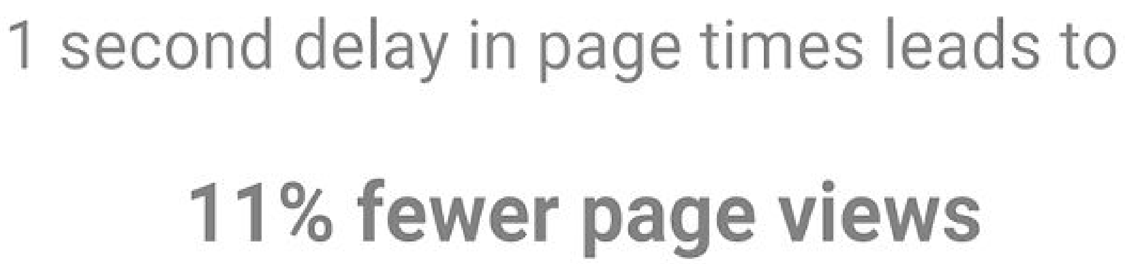
Using this pattern can bring dramatical improvement to page load.

[12] Source : https://developers.google.com/web/updates/2015/11/app-shell


3G
First visit
Repeat visit
https://app-shell.appspot.com/
App Shell
Index
https://app-shell.appspot.com/
App Shell
Index
2.5
0.8
Time to first meaningful paint
[13]

PWAs enhance first meaningful paint

We talk earlier about the bounce rate on Web application in regard of the page load. This pattern makes possible to address this issue and optimise businesses.


1 second delay in page times leads to

11% fewer page views

[14]

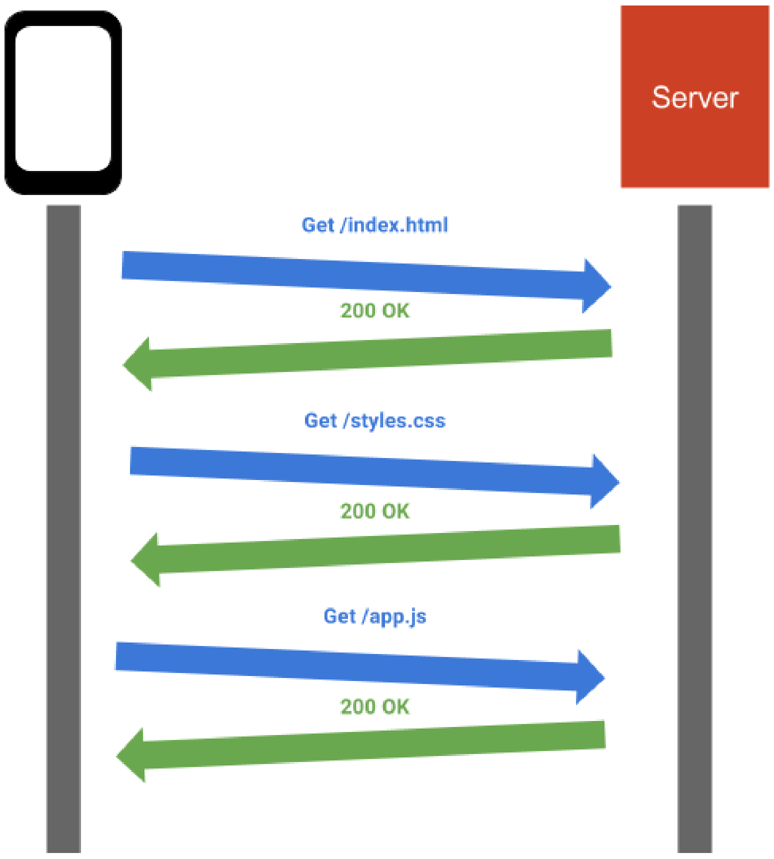
Delay rebound statistics

4. PRPL pattern

Achieving really fast first paint, is great but the rest of the application has to load fast as well. This pattern aim to structure and serve PWAs with great performances. By nature, the Web is build to fetch contents online and bring them back to the browser so it can process it and display the page to the user.

[13] Source : https://developers.google.com/web/updates/2015/11/app-shell
[14] Source : http://www.radware.com/Products/FastView/

This was simple when the Web was only static document with no external resources, we loaded the page with the text and we had the entire site. But nowadays, a web page is no longer just static texts, it contains images, CSS stylesheets, Javascript files, and even load via XHR new resources over and over.



HTTP requests round trip to the server

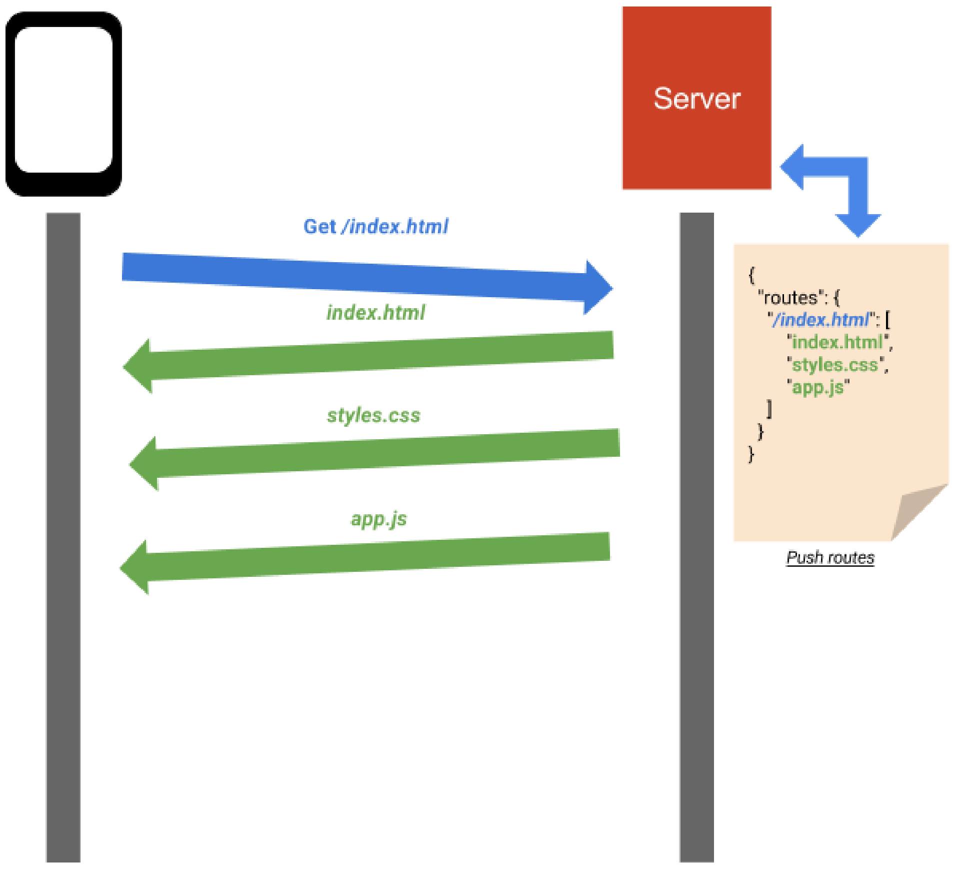
For each of those resources the browser have to create an HTTP request to call the server which then process it and response with the right document. The browser parse the response and identify the sub resources and then makes new HTTP requests to download the needed contents.

All those round trips to the server cause a huge delays to load the application. The PRPL pattern aim to address this issue.

PRPL, (pronounced purple), stand for :

- **Push** critical resources for the initial URL route.
- **Render** initial route.
- **Pre-cache** remaining routes.
- **Lazy-load** and create remaining routes on demand.

Each points of this pattern use all the latest innovation of the Web platform, but by targeting only few a them, it’s possible to enhance significatively the Web experience. PRPL is a must have not an absolute necessity.



```
{
  "routes": {
    "/index.html": [
      "index.html",
      "styles.css",
      "app.js"
    ]
  }
}
```

HTTP/2 server push

By using HTTP push from the HTTP/2 protocol, this pattern allow the server to send simultaneously sub resources to the browser so it has less requests to do.
By requesting at the time all the critical resources, PRPL use the App Shell pattern to early render the initial route.
The remaining necessary routes are prefetched user the new HTML5 preload API.
All other routes are then lazy-loaded using the fetch API.

Although PRPL may be used carefully because Server push force the browser to download content even if the are already cached.

5. Web App Manifest and Web App Install

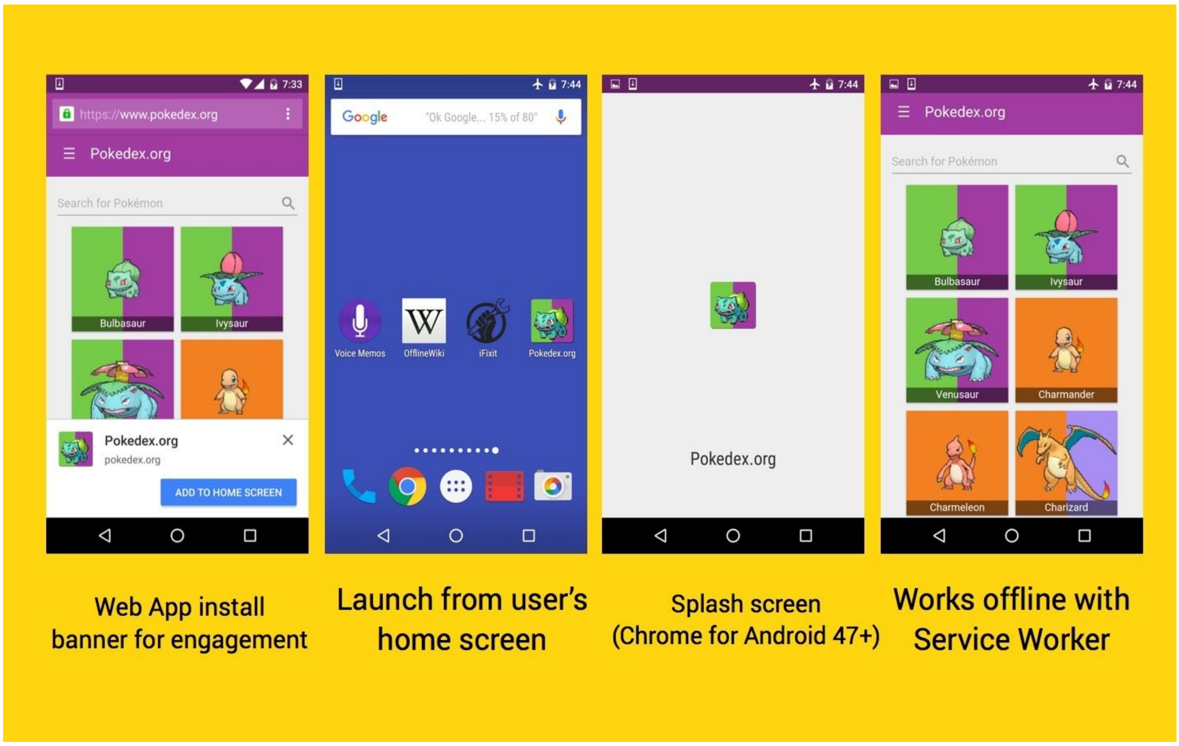
As all those new capabilities come in the Web platform, something was still missing to bring a more immersive user experience, leading developers to try out other solutions, such as

Apache Cordova containers, or React Native, to make their app closer to the system platform than browser did.

Browser vendors have a huge responsibility because they are the entry point to the Web and the link between developers and the system platform. As developers we asked really hard to get new capabilities that could make us full citizen of the application ecosystem. But this was not possible as long as the fast load and offline experience promises were broken.

The PWAs bring new hope by helping us keeping those promises to the user. Now we can claim to be first class citizen of the application ecosystem.
We said earlier that browsers are the entry points to the Web, in fact they are the only way, but as first class citizen of the app ecosystem, Web app should be launch just as native app are.
By nature, to access the Web user have to first launch the browser and then search for a website, even though we have URLs super powers, this isn't the best experience possible for the users.

The Web manifest, was created to bring this fully immersive experience to the Web platform, allowing us to perform app installation that add an icon on the home-screen, so the app can be launched without going through the browser. It also gives us an app like splash screen and access to device orientation, and them color, to feel just like a native application.



[15]

[15] Source : https://addyosmani.com/blog/getting-started-with-progressive-web-apps/

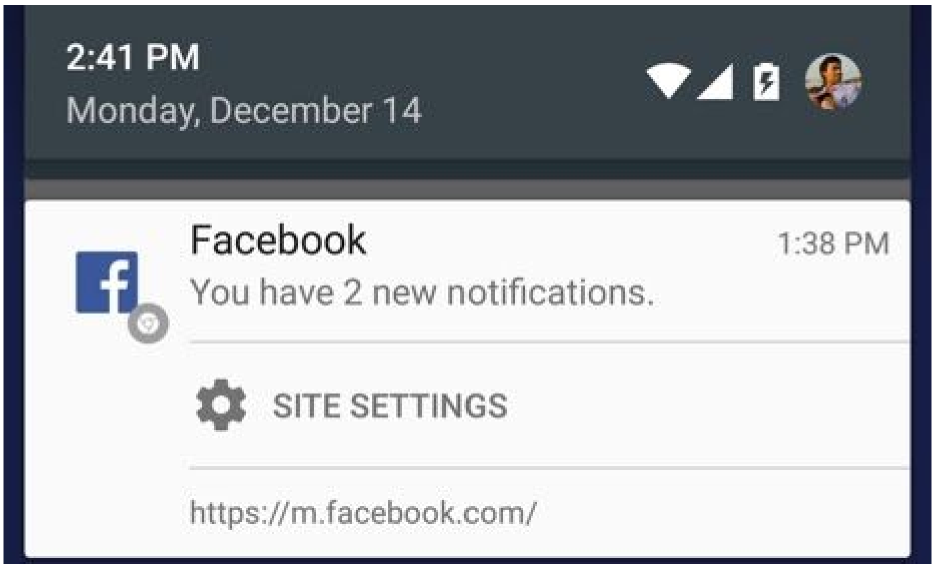
Add to home screen feature on Android.

As we are now citizen of the home screen, we need to kept our promises and fight hard to guarantee to the user that we won't fail.

6. Push notification and background tasks

User re-engagement is a real struggle for businesses, and the Web failed to help them, by being slow or not working when the user needed it the most. As we said earlier, thanks to Web App Manifest, users can launch their favorite Web apps, in a immersive mode, directly from their home screen as any apps, and as it a PWA, it will launch and work not regardless of the connectivity.

Push notification are a great mean to re-engage user to an application. Notification already exist on desktop browser but require the App to be open to display them to the user, which doesn't makes sense if what we what is to remind a user of our application.


2:41 PM
Monday, December 14
Facebook 1:38 PM
You have 2 new notifications.
SITE SETTINGS
https://m.facebook.com/

[16]

Facebook Web push notification

Thank to the Service Worker, it's now possible to run tasks in background, like fetching new contents for a website. By running in background, separately from the Web page, the Service Worker can listen for push notification events and display them to the user.

7. Browser support

[16] Source : https://addyosmani.com/blog/getting-started-with-progressive-web-apps/

As we go through all the key points of the Progressive Web Apps, we noticed that the Web platform, have an essential role to bring a compelling experience to the user.

But the truth is, not all browsers support PWAs. The major browser to start supporting those key feature was Google Chrome by enabling basic Service Worker support.

The Service Workers API arrived on Chrome's stable channel with version 40. This technology was already available since early December 2014 via Chrome 40 in beta, but the API now comes to the stable version. Opera quickly followed chrome by implementing it few days laters.
For Mozilla, Service Workers API was available one year later in January 2016. This feature is also available on Edge but it has to be activated in order to work.
Safari is the only one of the biggest browsers that still not have implemented it. It is “under consideration”.[17]

Chrome 42 inaugurates push notifications, allowing to deliver notifications to multiple clients, even so the client application is closed. Developers can use Google Cloud Messaging or other services to request a service worker script running in the background in the browser, separately from the web page. Once requested, the service worker can execute JavaScript code. Other brother vendor have since then enabled push notifications like FIrefox 44 and othen are working on fully implementing it like Microsoft Edge or Opera. As Safari does not support Service Worker which is a key factor for using Push Notifications. The later are not available on safari mobile.

Since 2015, Google gives the possibility, on Android, to push web apps via the homepage of Chrome. But this possibility being based on Android shortcuts, applications using this device could not appear so far as could be the native apps. The release of Chrome 57 changed that.

The browser will indeed integrate Progressive Web Apps more deeply into Android. For example, a newly "installed" PWA will appear in the list of applications on the device. It will also be present in the Android Settings. In addition, it will be able to communicate with other applications. Finally, its notifications will be handled in the same way as for other applications and will no longer appear as Chrome’s. For the time being, Chrome is the only browser to support this feature.

Google does not hide its ambition to bring the functioning of both types of applications closer together. For now, the editor focuses on friction reduction, that is, elements that can mislead the user. It was therefore not normal in this context that Progressive Web Apps behave differently and do not have the same level of integration.

[17]Source : https://jakearchibald.github.io/isserviceworkerready/

III. Constraint

Those innovations are great for each platform and the user will certainly benefits from it, without having to change the way he access those platforms before. But it will not be that easy for developers.

1. From Apps to instant Apps

Android Instant Apps are a real improvement for the high friction distribution model of the native platform. Since the beginning of the smartphone era, with Apple and others like Android, Blackberry or Windows Phone, the way users access applications hasn't really changed. The user has to connect to a store search for an application and then download megabytes of data and finally install the application on the device. Regardless of the size of the application that has kept growing, this model fail to adapt to the way of consuming application. Today we use few apps to do a lot of things but sometimes, needs a specific app to do one thing and struggle to get it done because download over mobile connectivity is awful and storage on the devices is not unlimited.
It’s the first time the platform try to change its distribution model, and try to streamline the process, to go to something more like a low friction distribution model powered by links.

But it will not be without pain for the developers who need to re-scaffold their entire application in order to deliver it in a way that the store can serve only a parts of this application. In other words, application developers need to code a new version of their application in order to take advantage of this feature, which can takes less than a day or weeks according to Google, depending how complex the application is.

This is a great attempt of native applications to try to solve the distribution model friction. But it feels like it's moving the friction from the users, that were needed to go through a store to access apps, to developers who need to recode their apps.

Privileging user experience is always a good bet, but having an over complicated development stack for developers can be worse because of the lack of control over this huge stack, or even discourage developers to engage in this process.
That is why innovation for those platforms have to be seamless for user but for developers as well, in order to encourage them to transition to it.

An other problem to solve for Instant Apps, is the security level. As native apps can have direct access to the smartphone APIs and resources, using app that seamlessly lay down on the user smartphone can be very dangerous. Even though, Android apps require authorization before letting apps accessing the smartphone’s APIs, as its will only need a link to download and actually run an app on the device, this can be a difficult problem to solve in order to gain user trust.

But the most important and more obvious constraint to Android Instant Apps is that, as its name let say, it's limited to the Android platform, so all IOs apps users can't benefit from it.

As natives apps does not lay on a unified platform, it's more difficult to try to solve the app distribution model, if both parties are not on the same page.

2. From Web Apps to Progressive Web Apps

This struggle is also true for Web applications, and maybe more difficult because of the multiples platforms that power the Web, browsers. In fact there are even more browsers than mobile platforms and none of them integrate APIs in the same way, which result in what we call the browser vendor dance. Where, developers have to specify each platform.

```
.example {
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
}
```

Because of this nonuniform platforms, implementing features in Web app has been a real nightmare for developers for a long time. Even though, most browser vendor are now working together to ship the same APIs, they do not priorities them the same way.

This disparity led developers to either, chose to implement a feature for certain browser where it works and blame the user for using another one, or to use huge frameworks library to ties the gap between browsers, which in the two case is bad the user experience.

But sometime even frameworks can't replace lacking features from the browsers. This is the case for the Service Worker API that is not implemented in Safari mobile browser. Even though, Progressive Web Apps are about overall great user experience with graceful degradation, the lack of will from Apple to implement Service Worker make Progressive Web Apps an Android thing and discourage developers to go to it because they will not hit IOs Safari users.

Converting a simple Web app to a PWA, requires developers not to entirely recreate their app as for Android Instant App, but to rethink the way they serve content to their user and of possible to add a Service Worker to handle offline requests.

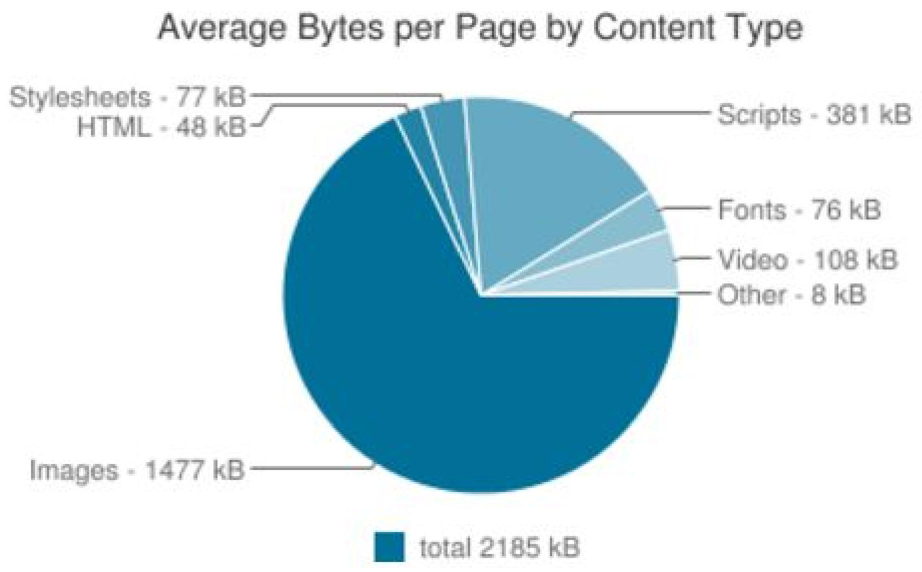
III. Our solution

In regards of our analyse, the Web have now all the necessary tools to provide to the users the best user experience possible with a low friction platform that is accessible to all users around the world. But this is only useful if application producers and developers chose to use all those new capabilities.

Sadly enough, the Web platform is still considered as an unstandardized and immature platform, and Web developers themselves do not consider crucial to fight for it to be trusted as a robust platform.

The plurality of JavaScript modules that are created every day, is often used to show how unstable the platform is, and how impossible it is for developers to stay up to date, leading them to not pay attention of important innovations made for the platform. We couldn't blame them, because it’s true that the number of new capabilities proposed by libraries developers or browsers vendors is overwhelming. But this is what make the Web one the most successful open platform, because developers are free to try and contribute to it evolution.

Earlier in this paper we briefly talk about the Web page bloat crisis that the Web have to face. As said before, the overall content downloaded for a Web page is way too high in regard of the connection capacity. Where most of the people are not in a 4G LTE connection, more than 2Mb of data are transferred for a single page. This is due, for instance, to the amount of uncompressed images send to the client, but after images come the scripts.



[18]

The script represent nearly three times all other downloaded content in size. In fact, the size of downloaded script has not ceased to increase since it appeared on the platform.

Over the years, we've seen a common pattern that is transferring all the logic from the server side to the client side using the server only to fetch data.

This was great because it gives more flexibility to developers to manage their applications and gives them freedom from the obligatory server round trip. But this freedom have a cost, and while deporting all the logic from the server to the client, front-end developers need to create more robusts scripts to handle all those logics. This result in the creation of multiple heavy frameworks that take care of this logic.

This heavy script load is one the major issue that ironically tears down mobile Web experience on mobile while it was supposed to enhance it.

Framework / Libraries	Size (*minified*)	Size (*Gzip*)	2G (*0.1Mb/s*)
jQuery (library)	93Kb	32Kb	~ 2s
AngularJS	566Kb	111Kb	~ 9s
React (library)	133Kb	39Kb	~ 3s
Vue.js	63Kb	23Kb	~ 1s

[18] Source : http://mobile.httparchive.org/interesting.php#bytesperpage

In this table we can see how heavy those libraries and Frameworks can be. AngularJS, one the most popular modern Frameworks take nearly 9 seconds to be download over an average slow connection. And it without any of its thousand of modules.

We are not saying that frameworks or libraries are bad things, they are powerful tools to help create more robust applications but the cost of using them careless is too high to not pay attention.

After our research and speaking with dozens of fellow developers we came up with one conviction. **No one wants to make bad user experience and application that load slowly, it's because developers often doesn't know how to do better that they are doing mistakes.**

In this regard, part of our solution to face this issue is pedagogical, we have to raise awareness of developers.

A. Raise awareness

To do so, we imagine a simple quiz game that led them through all the possible mistakes developers are often doing while developing Web application, without knowing it. In order for them to realize it's possible to do better.

That quiz will also be helpful for us to gather metrics about developers habits to adjust our communication and subjects we communicate about. And then help developers to avoid common mistakes.

Sadly, one common belief is that, doing better also mean doing more complicated. Ultimately it can be true, but there are small optimizations that can be done without any cost.
For instance, as we pointed out earlier, browsers are the platforms that power the Web, and the more those platforms become powerful, the less developers need to use heavy frameworks to bring lacking functionality to the platform. If a feature is missing from one specific platform, using polyfills instead of a whole library or frameworks is crucial because it doesn't force platforms that currently support this feature to download unnecessary code. The scriptability of our platform has astonishingly evolved, and developers need to take that in count.

This is the kind of pedagogical contents we intend to deliver to as many developers and applications producers as we can through articles, videos podcasts or talks. The final thought is for us to create an organisation that aim to help developers and others organisation to improve their development by providing content such as articles, tools or services. This what we already intended to do by writing this paper, but we have to do more than that.

B. Providing tools

As developers, we can't exhort others to do what we aren't doing ourselves, so in order to show the path, we are committed to create open-sourced rich applications examples and tools with all the best practices to deliver good user experiences. So developers can take the code and play with it to make it their own.

The developer experience, is often compared against the user experience, but good user experience doesn't necessarily mean a bad developers experience.

In fact, as students in engineering school, our main focus was to bring developers the best developers experience while giving them the canvas to create a rich user experience. So we intended to create a new kind of CMS, a CMS to create Progressive Web App Content Management Systems.

On the Web, the majority of websites are powered by CMS, in fact, only a single of them represents nearly 60% of internet websites around the world.

Major CMS often offer a bad user experience and an even more bad developers experience. Those CMS are not created for mobile and yet are used for contents websites, which, are the most used on mobile devices.

Through talks with other developers, we discover that even so CMS are supposed to be modular, developers often consider that it's too hard to add new features to the application, due to it complex architecture. So adding new features such as offline support is unthinkable.

We want to create a powerful and flexible tool that help developers create powerful content management systems while using all the capabilities of Progressive Web Apps such as offline support and many more.
Raising awareness and providing great tools is the our way to revolutionise the development of Web applications.

Conclusion

At the end of this research on the future of the web based on the progressive web apps, some questions introduced in the introduction, and then developed throughout the theoretical part, found answers. Developers often compare native application against Web application, but through this paper we have seen that, this is not the right way to thinks about it. It mostly depends on what you want to create.

For instance, if you target the Next Billion Users, which are located in developing countries, Progressives Web Apps are the way to go since data consumption is such a sensitive matter in these regions.
The same applies if your application aim to target only desktop users or if you don't need Android-specific native features. It's more about what your and user need in matter of experience than which technology is better that the other and where are located all of our users.

We have to keep in mind that providing a better user experience for people in others, such as developing ones, ultimately will provide a better user experience to all our user, even in developed countries.

It is also undeniable that acquiring users is much easier on the web. On average, 26,4% of users that visit your app store page will install your app and according to a study by ComScore, the top 1000 mobile websites receive 270% more monthly visitor than the top 1000 native apps.

Looking all these facts, we could say that Progressive Web Apps are a better approach than native applications, in order to deliver a rich experience to the end user.

But in fact, it is not that simple. Native applications have access to features that Progressive Web Apps currently lacks such as telephony features, low level access to some hardware features, alarms that makes them still really relevant.

The goal for Progressive Web apps is now to expand these capabilities to cover the most critical cases and when it will append, we will ask ourselves what about the future of native applications.

Annexes