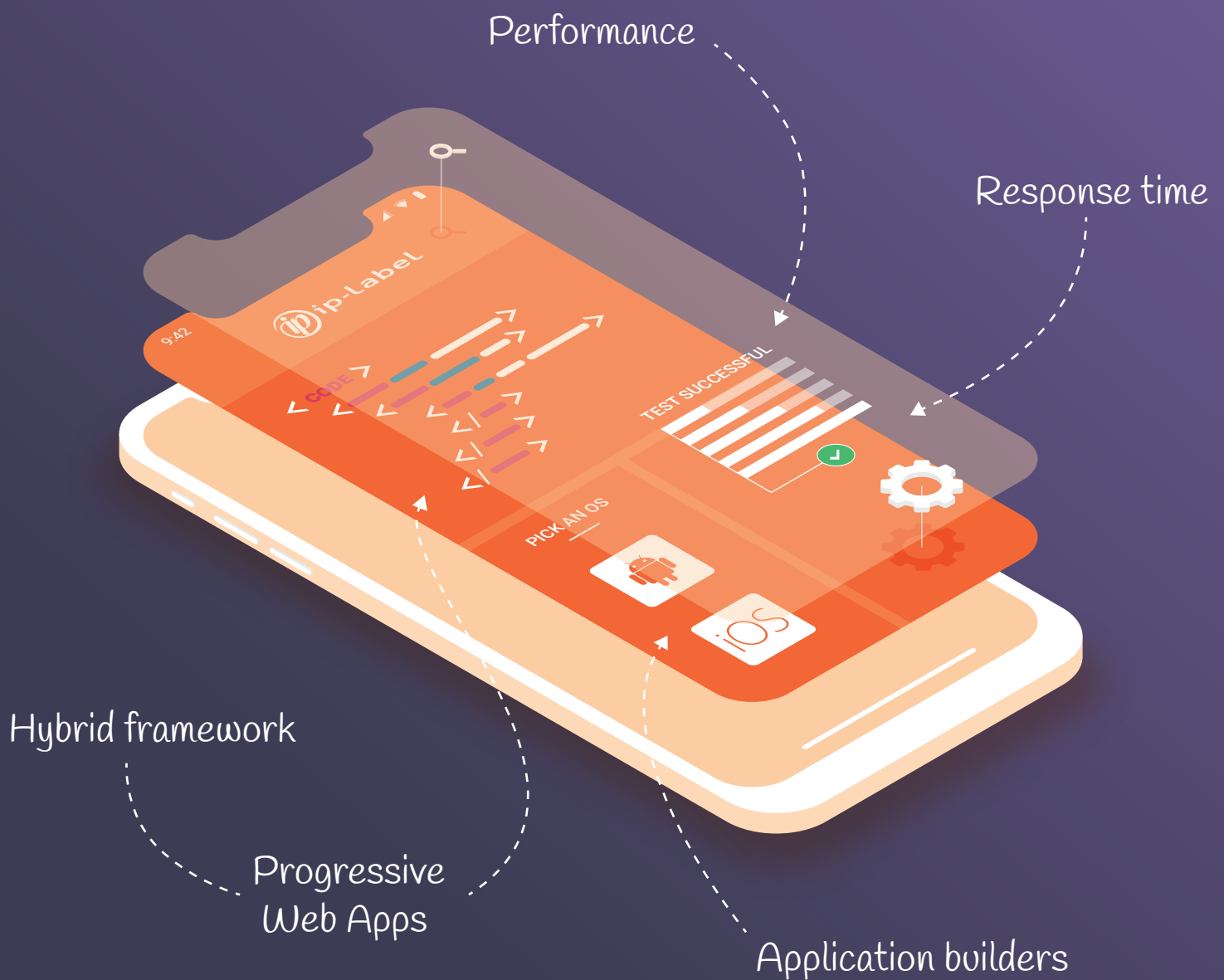


Delivering high mobile performance to your users

Design strategy, performance criteria, user experience, and more !



Contents

1	Choosing the right design strategy	04-12
	Defining a mobile project	05
	Mobile-dedicated or responsive websites	06
	Mobile applications	08
	summary of design choices	12
2	Developing a mobile application	13-19
	Choosing an environment (IDE)	14
	Choosing a framework	15
3	Testing and acceptance	20-24
	Unit tests	21
	Functional testing	22
	Integration testing	23
	Load testing	24
4	Deploying an application to production	25-31
	App stores and markets	26
	Publishing a mobile application	26
	Statistical and analytical monitoring	27
5	Where does Performance fit into mobile usages?	32-42
	Mobile website performance	33
	Mobile application performance	38
	Active + passive monitoring: a winning strategy for mobile usages	41

Introduction

Mobility: a major issue in digital transformation

The proliferation of mobile devices and usages (smartphones, tablets, laptops...) has significantly changed the new technologies marketplace. Mobility is an essential component in the digital transformation of any enterprise and any organization, regardless of its primary activity (commerce, logistics, education, transportation, healthcare, etc.).

These past years have been characterized by a huge increase in mobile purchases, particularly in Europe. Mobile is carving out a place of its own with respect to the PC.

Furthermore, the number of Google searches made on mobile devices exceeded the number of searches on fixed PCs. Google announced that it was taking mobile performance into account in its search result rankings starting in July 2018, which means that performance indicators should be planned for as early on as possible in development as well as in production.

This said, **bringing a mobile project to fruition can be a tricky business if you do not take account of all of the factors of success.**

If the performance of mobile services is deficient, user experience is consequently unfavorable. This can greatly harm the brand image, productivity, or even lead to lower revenues. **User experience is the basis of the success of a mobile project.**

How can you best plan your mobile project? What strategies can you adopt? In what way is performance a foundation of a successful mobile project? Discover our best practices in this white paper.



N°1

Design |

Choosing the right design strategy

Defining a mobile project

Before embarking on any project, the first requirement is to specify the precise aims and target of the project, and to define the technical skills needed to achieve the outcome.

These objectives may include, in order of priority or maturity:

1. **Developing a high-performing mobile web presence** on all types of devices to attract new users/customers. For instance, on social networks or search engines, or via advertising campaigns. The fact is that mobile web growth is accelerating inevitably year after year.
2. **Strengthening user loyalty.** For this aim, one important step is to roll out a mobile application that matches the core business.
3. **Defining a new business model:** digital subscription via an app store, integrated purchase functionality in the application, etc.

In all cases, the need for specific **technical skills** is another crucial aspect to address and secure: Web/HTML/JavaScript developers, or developers with dedicated iOS/Android mobile skills.

The decision to invest in a website adapted to mobile usages, a mobile application, or both at the same time is a choice that must be made in accordance with the overall aims of the project, the available skills and human resources, and the budget that can be devoted to the project.

What are the advantages and disadvantages of a mobile website compared to a mobile app? Keep reading for a few things to take into consideration.

Mobile-dedicated or responsive websites

Depending on the needs of the organization and its internal and external customers, mobile websites may be either the sole mobile point of access or an additional resource supplementary to mobile applications.

The first choice of mobile website design is whether to create a mobile-dedicated site or a responsive site:

- **Mobile-dedicated:** for this option, mobile is considered target number 1, with templates and JS/css resources adapted to mobile, in addition to content with resources, including images, that are oriented toward mobile usages.
- **Responsive:** this type of site implements the same resource basis (in particular CSS, but JavaScript as well) for desktop and mobile websites.

Mobile-dedicated websites

A mobile website is a website dedicated to use on a tablet or smartphone. The content and functionalities of this type of website are optimized to take account of the specificities of mobile devices.

The pages are specially designed for ease of use on a mobile device, including how it displays on the screen, tactile functionalities, menus, etc.



Pros

The mobile-dedicated website provides fast display and is easy to read on mobile devices. Technically, this is one of the simplest ways to build a presence on the mobile web.



Cons

There is no compatibility with desktop websites (different URL). Furthermore, mobile websites must be developed and managed separately from desktop versions of the site in order to ensure a web presence that is adapted to all devices. This means greater costs for creating, maintaining, and updating the mobile website.

In terms of **performance**, the main points to pay attention to are not very different from those that improve the performance of any website: the total volume of resources loaded on the network and their order of loading, compression, the number of HTTP requests, and reducing processing on the client side when running JavaScripts and third-party widgets.

To sum up, the site's design must facilitate fast display of all pages and good performance of all services for users in a mobile context.

Below, in the chapter on frameworks, we will take a look at a variant of the mobile-dedicated website centered on Google's AMP initiative.

Adaptive or responsive design websites

A 'responsive design' website is a single site that displays on all types of terminal (tablet, smartphone, computer screen).

This type of site adjusts automatically to the size of the screen of the device used. Such automatic adaptation of the display uses the information provided by the device when the page is created or formatted (CSS 'media queries').

Consequently, this increases the display options that developers have to manage, and must be considered at the outset.

For instance, it is possible to display or hide some elements, reduce the height of menus, transform columns into rows for vertical display, etc.

While managing a different layout for each screen size makes site design more complex, market frameworks (or content management templates) facilitate this task.



Pros

Costs are controlled because this is a single site administered at the same point. The same developers with HTML/CSS/JavaScript skills can adjust existing code for mobile display.



Cons

The display is adapted to the screen in question, but the content (videos, features, menus) are not necessarily optimized for mobile situations. This may lead to the problems of how much it weighs down the network and of the resources needed for the rendering (mainly CPU). Nevertheless, such assets can be adapted to the terminal, for example by not loading certain parts of the site, like backgrounds or image carousels, etc.

Furthermore, each page must be tested on the various devices to check whether they display content correctly, and this lengthens development time.

Mobile applications

Going “native”

Native is a term that is used when the application is developed with the tools and languages proposed by the operating system specific to a platform; for example, Java for Android platforms, Objective-C and Swift for iOS platforms.



Pros

Native is a seal of performance, as loading is faster because it is done locally. What's more, the application is accessible without a connection. Native development allows the application to be adapted more quickly depending on the type of platform (iOS device or Android). It is also indexed better thanks to downloading platforms.

In short, the advantages are:

- performance
- ensured UX
- push notifications
- full screen mode



Cons

If the need is to develop a cross-platform application, you must maintain as many source codes as there are target platforms. Each platform has its own specificities, and this can make development and updates expensive.

Developing an application in Android or iOS native environments requires specific technical skills. Still, in the interest of saving time, you can rely on frameworks to delegate certain functionalities, such as hosting or the database.

It costs more to develop an application for Android and iOS devices at the same time, in a programming language specific to each of the platforms.

Lastly, to conclude with a recent trend, according to the latest TIOBE study and index (see <https://www.tiobe.com/tiobe-index/>), SWIFT may be losing ground to hybrid frameworks, which we will take a look at in the next chapter.

The hybrid approach

A hybrid application is developed with elements of HTML5 combined with native elements (for example in the form of a library). A hybrid framework produces, in

a 'build', a native application which can then be released on the AppStore (iOS) and Google Play (Android) downloading platforms.



Pros

The time it takes to develop a hybrid application is much shorter than for native apps because the source code is common to each of the target platforms. Existing skills in HTML/JavaScript are transferrable to hybrid development. Maintenance is likewise simpler because revisions are made on a single version.



Cons

In terms of performance, hybrid applications may not be as fast or stable because they are less specifically adapted to each platform. Furthermore, the build is not the same on all downloading platforms, and problems may occur during updates.

There are also compatibility constraints between the hybrid framework and the underlying specialized building blocks (for example, Angular/JavaScript).

We'll return to the various hybrid frameworks (IONIC, Flutter, React native...) later on, in the section about how to choose a framework.

Progressive Web Apps

Progressive web apps, or PWAs, are based on SPA-type web pages (single-page app) to provide a user experience close to that of a native mobile application.

<https://developers.google.com/web/progressive-web-apps/>

The key characteristics:

- icon on the home screen (favorite)
- immersive full-screen experience
- push notifications
- immediate loading, independently of the state of the network
- fast and smooth: resources conceived for mobile from the start

Technical requirements:

- UI components designed with these constraints in mind
- manifest must be declared
- use of service workers (for the local cache and offline operation) and local storage

Among their 'native app' characteristics, PWAs allow use of the telephone's capabilities (notifications, microphone, camera, GPS, etc.) and loads immediately, independently of the state of the network

(offline operation) by calling on a local proxy ('service workers') which must be instrumented by the developer.

As for their ‘web’ characteristics, PWAs adjust their display to any terminal, mobile or desktop. Moreover, as they are based on web technologies (HTML, JavaScript), no native compiled file (APK ou IPA) is produced. PWAs are not dependent on any downloading platform for their release, and

– just like viewing a website – access to the interaction phase is very fast. Indexing by search engines is done in the same way as for any website; therefore SEO strategies can be used.

Pros

The advantages of PWAs are the same as those for native mobile applications in addition to those of mobile websites. Some people feel that PWAs are the only sustainable option at present.

Pros:

- push notifications
- offline
- icon
- full screen
- instantaneous loading

Cons

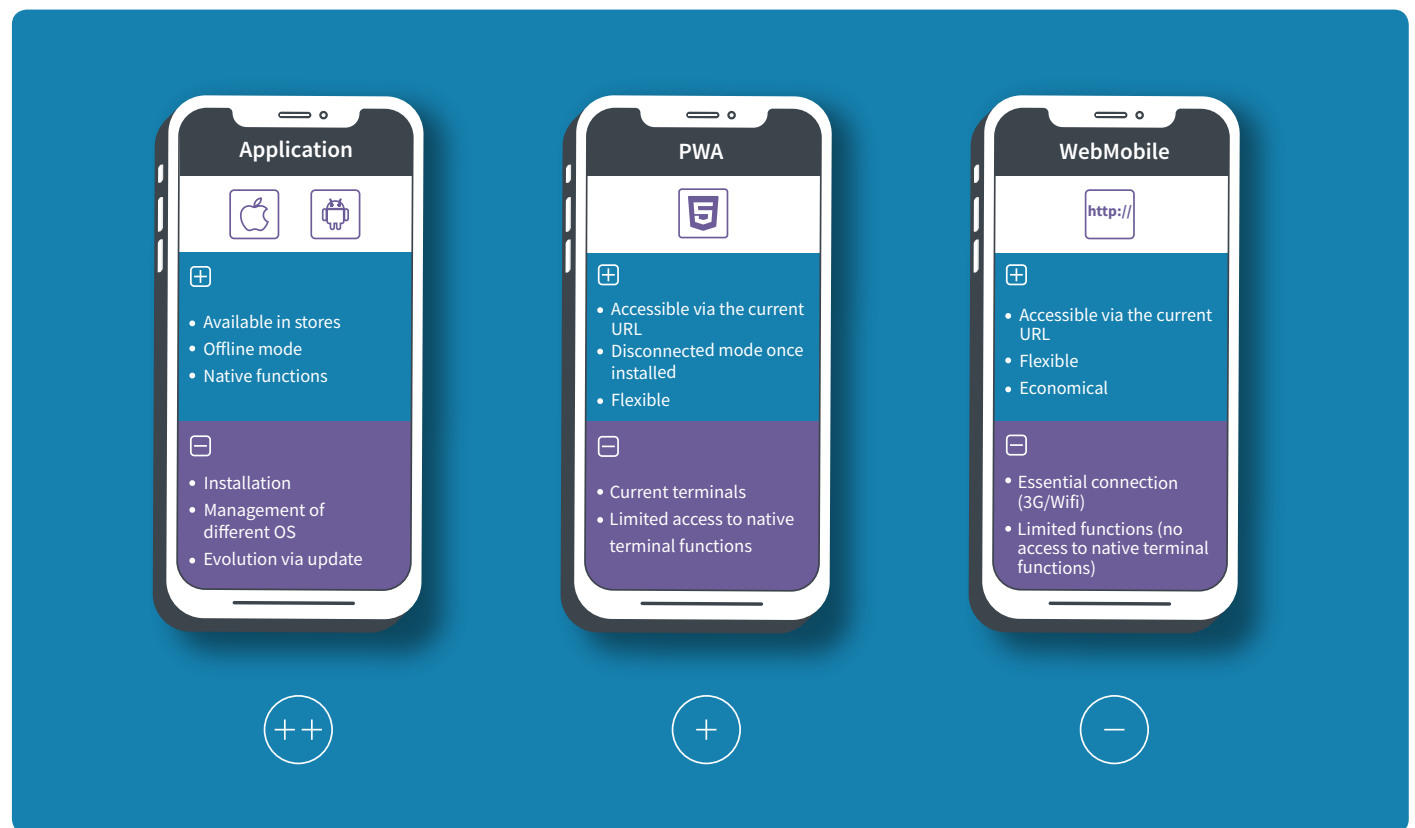
There is no major disadvantage to PWAs; there are only a few technical requirements such as declaring the manifest, use of service workers, and local storage. At the same time, all frameworks can help create high-performing PWAs by facilitating the use of service workers and local storage.

Note that this technology does not function as fully on iOS as on Android/Google Chrome. For example, some features are not enabled in Safari. On the other hand, there is talk of better performance on Apple, with

reference to greater involvement, etc., and it appears that Apple may be increasingly interested in PWAs.

Summary of design choices

The pros and cons of design choices are summarized below:



Frameworks



Nº2

Development |

Developing a mobile application

Choosing an environment (IDE)

The first step in the mobile development process consists of choosing the platform on which to deploy the product or mobile application.

Nearly all applications are developed for Google Android or Apple iOS. Developers must nevertheless deal with a growing number of development environments (IDE for 'integrated development environment'). The choice of an IDE that is 'official' or 'adopted' by mobile operating system vendors enables access to all of the functionalities provided by those vendors, including product updates, help, and a rather knowledgeable community of developers.

Some of the best known IDEs are:



Xcode for iOS

Languages: Objective-C / Swift

Xcode is the native, official IDE from Apple for the iOS mobile operating system. It comes with a full software suite for developers.

Xcode development is done essentially in Objective-C or Swift (available only on Mac OS).



Android Studio for Android

Languages: Java / Kotlin

Google provides its official, supported IDE called Android Studio for the Android mobile operating system. This environment replaced Eclipse in 2014. Development is done in this IDE essentially in Java. More

recently, in May 2017, Google adopted Kotlin, a language created by JetBrains, as the second development language for its Android OS.



Netbeans - Android

Languages: Java / Kotlin

Like Eclipse, Netbeans is an IDE that uses Java. A Netbeans Mobile package is available, providing a full software suit for developing for Android. Netbeans is an environment which, depending on its version, allows development on different

types of technologies and platforms like web or mobile. As with Eclipse, developers need a plug-in for Android.

Choosing a framework

A framework is a set of building blocks or reusable software components that underlie applications and software. The purpose of a framework is to simplify work for developers. The main advantages of a framework are the **reuse of code** and the **standardization** of software development.

There are hundreds of frameworks covering most programming languages. Developers sometimes use a combination of frameworks to enrich graphics and for the architecture

Frameworks for developing hybrid or native mobile apps

A variety of frameworks exist for building cross-platform or native mobile applications. Each framework offers its own set of advantages and limitations.

Some of the best known frameworks for developing hybrid or native mobile applications:



Xamarin (hybrid)

Xamarin is an open source project acquired by Microsoft. Mobile applications built with Xamarin are programmed with libraries based on .NET. The solution also lets you

call code in Objective-C, Java, or C/C++. Xamarin contains links with the SDKs of iOS and Android platforms.



Cordova (hybrid)

Cordova, formerly known as PhoneGap, is the main framework for developing hybrid apps. Developing Cordova applications is now very simple and natively managed by many IDEs including Visual Studio 2015.

Cordova's architecture is modular, based on plug-ins which bridge between JavaScript and the native implementation specific to each platform.

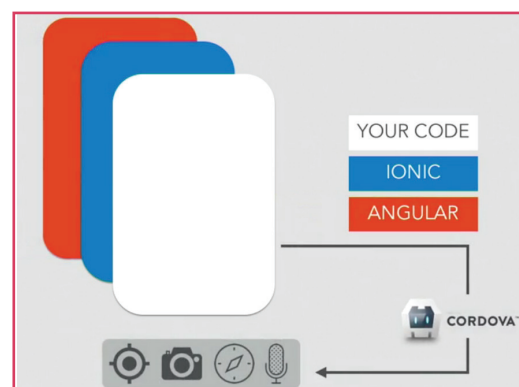


Ionic

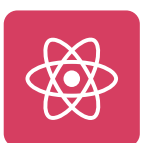
Ionic allows developers to reuse web skills (HTML, CSS and JavaScript) to develop native apps. The "Ionic native" version integrates

Cordova for access to the smartphones hardware elements (camera, etc.) as well as social media plug-ins (Instagram, etc.).

Ionic also uses Angular, an advanced JavaScript web framework, for modeling exchanges or refreshing application data.



Source: lynda.com



React Native (hybrid)

React is Facebook's web development solution, which has been adapted to generate mobile applications. It lets

developers to capitalize on general web skills (HTML, JavaScript) to generate native apps (IPA or APK).



Google flutter (hybrid)

Flutter is sometimes seen as Google's response to Facebook for mobile app development, but it is based on its own

programming language. Development with Flutter is based on the use of a 'canvas', where application objects can be drawn.

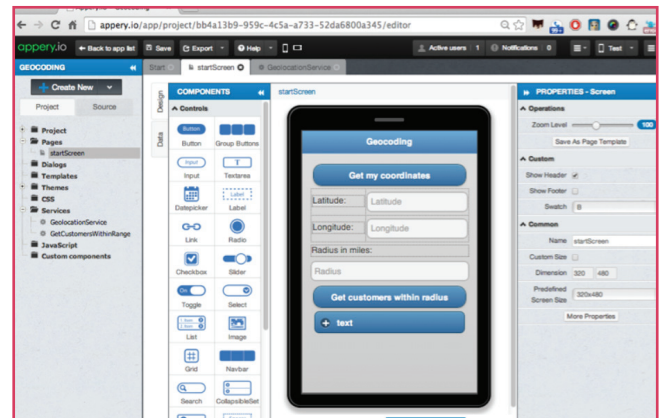
Application builders

There are also solutions for quick development of simple mobile applications (hybrid apps or responsive web apps) using a visual interface (drag & drop). These application builders are also called “Rapid Mobile Application Development” (RMAD) programming tools.

These solutions supply predefined themes, but access to native functions must still be coded ‘natively’. One example of a RMAD environment, also available in the cloud, is shown here:

Some of the most widely known RMAD tools are:

1. Appery.io
2. Appcelerator
3. AppInstitute



Source: appery.io

Cloud services framework



Firestore

Firebase is a particular framework which includes ‘client’ components (services for mobile apps, like messaging, advertising, etc.) and ‘back-end’ components (Google Cloud) like databases and hosting libraries on a CDN. Firebase can be used either with a native language or integrated into a hybrid app framework.

In addition to providing development building blocks, Firebase offers some additional features like an advertising network, monitoring of user behaviors (analytics), performance tracking, etc. The application’s GUI, however, is up to the developer to take care of.

Web frameworks

As mentioned above, **PWAs** (progressive web apps) appear to be the wave of the future.

Web frameworks can be used to create high-performing PWAs, as can native JavaScript. The differences lay mainly in the templates and components. These frameworks can help in setting up a PWA that provides:

- offline operation with service workers and local storage
- responsive UI components with tactile functionality

Some popular PWA-compatible web frameworks are:



Polymer

This framework was designed with progressive web apps in mind. The default app template and the Iron and paper components let you create a web app from scratch intuitively.



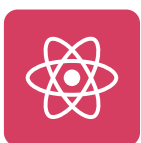
Vue.js

This framework proposes an application skeleton with everything you need to design a PWA.



Angular

When service workers are declared in the app's config file, angular-cli automatically creates the structure needed for deploying a PWA.



React

Many “boilerplate” exist, but nothing is proposed by default with the framework.

Accelerated Mobile Page (AMP)



AMP

Lastly, a full-feature framework for mobile-dedicated websites is the AMP (Accelerated Mobile Page) initiative from Google. AMP offers tools for editing the code of web pages to optimize them for mobile.

This is done by providing:

- the language (AMP HTML) and the AMP JS library, which comes with the AMP Validator
- dissemination via Google AMP Cache

Using these tools is a very effective way of making web pages perform fast and well for mobile usage. Furthermore, these speedy AMP mobile web pages appear to enjoy favorable rankings in Google and Bing search results (which would be in keeping with Google's 'mobile first' policy).

Using AMP is relatively simple and can yield excellent results. There are also real-user performance analysis libraries integrated into the AMP project.

At the same time, it is worth bearing in mind that detractors of the AMP project express concerns:

- over the significant technical constraints concerning the use of CSS and third-party content.
- that the immediate gain in performance comes from limiting standard HTML and resource-loading, which can be done without AMP.
- about using a product which has its own language instead of programming in accordance with accepted standards (W3C).
- that the CDN service offered by Google is free, but not very different from any other CDN, and therefore why voluntarily confine mobile services distribution to Google only?

In conclusion, the AMP initiative is not promoting the creation of a standard, but rather is advocating the use of a product directly related to Google (even if the project itself is open source).



N°3 Testing and acceptance |

Testing and acceptance

The complexity of developing mobile applications has grown over time, making the release of error-free applications nearly impossible. The quality of mobile application testing and validation processes has a direct effect on the app's success with its users.

Several different types of tests may be used at different moments of the development cycle:

1. **Unit tests** are conducted by developers to test their code during the development process.
2. **Functional testing** is performed on an application feature or features, taking different contexts into account (screen size, etc.). **Regression testing** during an update belongs to this category, as does **usability testing**, where a feature may work, but still be difficult to grasp for the user.
3. **Integration testing**
4. **Load testing**



Developing applications and websites for mobile usages makes test scenarios more complex because they must allow for:

1. Variable screen sizes: for instance, 1920 x 1080, 1368 x 768, etc.
2. “Portrait” or “landscape” page orientation
3. The type of device (and type of browser for a web application): iOS, Android, etc.

Unit tests

Unit tests are conducted by developers to make sure that a certain block of code (the ‘unit’), or part of the software or application (a ‘module’), performs the way it should.

These tests are intended to help the developer find bugs faster and facilitate source code maintenance and documentation.

Functional testing

Functional testing serves to check whether the application works the way it is expected to in various situations (different devices, OS versions, screen sizes, etc.). Such testing is imperative to make sure the quality is sufficient for the people using the application.

In practice, functional tests are often carried out by a **panel of users**. There are crowdsources solutions to achieve wider coverage (for instance, <https://www.stardust-testing.com>).

Beyond such user panels, it is important to have a **private distribution tool**, like an internal app store, to control distribution to a group of testers. This is advantageous in cases where there are many updates.

For Android, distribution is not limited, so that any user who has a compiled version (.APK) can install it on his or her telephone.

For iOS, a test device must be connected to Xcode or the application must be signed with the ID of a target device. This is quite cumbersome, but there are tools to facilitate the process, such as the TestFlight app. There are several alternatives to TestFlight, including **HockeyApp** (Microsoft) and **Crashlytics Beta**.

Test automation

Beyond the functional aspects, it is important to test the robustness and stability of the application by letting it operate continuously over a sufficiently long period.

This is where test automation comes in. To implement automated tests, the open-source library Appium, is very useful, and can be integrated into Java scripts, Python scripts, and scripts in other languages.

In a mobile continuous integration project, these scripts are synchronized with compilation using tools such as Jenkins.

An example of a script automated with Appium is shown below:

```
# Start App
#-----
scen.startStep(30000, "Start App", device)
device.startApp("Datametrie App", "com.iplabel.datametrie")
merr = ""Wait for app started""
login = device.element(xpath="//XCUIElementTypeApplication[@name=
scen.stopStep()
```

Source: ip-label RIALity mobile

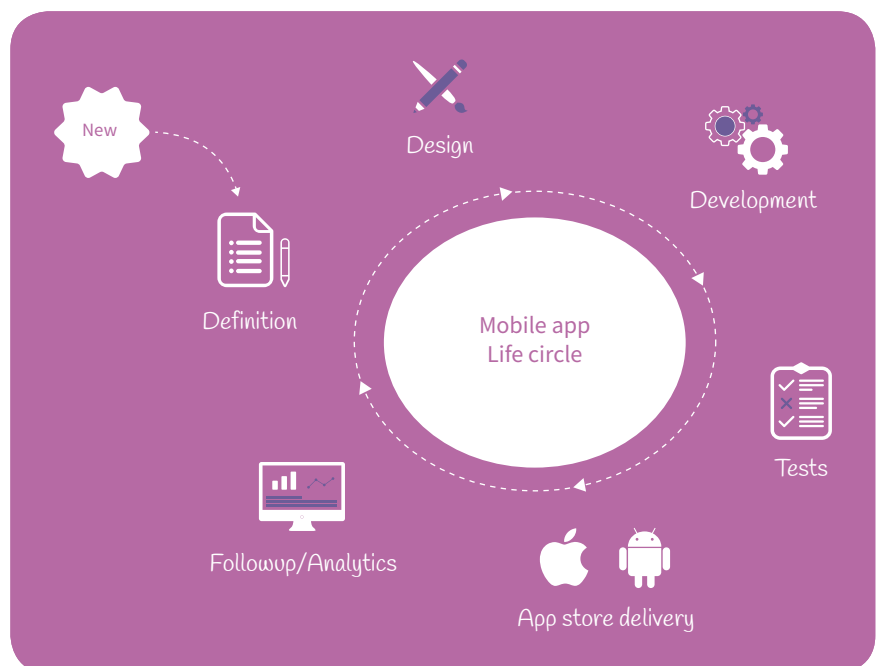
Integration testing

Integration testing can take place when the building blocks of the application are put together for the:

- backend/API
- front-end
- mobile application

The integration platform used is different from the development platform and nearly the same as the production platform.

When the production cycle integrates compilation and application deployment with automated test scripts, it is called **continuous integration**. The benefit of continuous integration is that it allows quality to be checked quickly whenever any change is made to the application (source code).



Tools are available (Jenkins, Bamboo...) for setting up continuous integration for the critical tasks of:

- compilation automation
- automating deployment to the test infrastructure
- automating validation testing, for example with a framework like Appium

When the application is rolled out to production (in an app market), it is still necessary to monitor the quality that it delivers to users. For this phase, the tools and techniques are described in part 4 of this white paper.

Load testing

Load testing or stress testing is used, either in pre-production or more frequently in production for mobile apps and websites, to check how the application or website reacts when a large number of users simultaneously connect and perform transactions on it.

There are a number of objectives, among which are to:

- to know what the platform's maximum capacity is
- identify bottlenecks in order to fix them

In the case of a web application, load testing can be conducted from an infrastructure located on the network core to simulate HTTP requests, if necessary with the mobile 'user agent'. For example, ip-label's Datalimit load testing service has its own load injection infrastructure, which is provided for the duration of the load testing campaign.

For mobile applications, the infrastructure needed for emulating a large number of mobile 'client' devices can be costly. In such cases, the APIs within the application can be solicited to saturate the back-end with HTTP requests, as for a website.



N°4

Publishing an app |

Deploying an application to production

App stores and markets

To release an application to the whole world or to a specific region, the creators of mobile operating systems such as Apple and Google have created the notion of app markets or app stores.

Apple introduced its market, the App Store, in 2008. It showcases several million

applications and has processed hundreds of billions of app downloads.

In 2012 Google introduced its store, first called Android Market, and now Google Play. This market contains several million applications, and has provided billions of application downloads.



App markets can manage payments and subscriptions to applications, as well as distribute updates. They also have usage tracking features with real-user statistics.

These management functionalities imply a compensation for Apple or Google in return.

Publishing a mobile application

Publishing an application makes it visible and downloadable for all users of that store. A number of verifications and restrictions are imposed by Google and Apple before your application can be accepted and posted in their stores.

Android

Google has no validation process in place for any application posted. This is why it doesn't take very long – about 2 hours – for apps to become visible on Google Play. Google checks licenses to ensure that copyright is enforced. There is, however,

a system called 'Bouncer', which scans all of the applications posted in the store to detect and delete those which raise security issues. Regarding coding standards, Google imposes no specific rules.

To publish an application on Google Play, a developer must enroll in the Google Developer program, a subscription which

costs USD 25 per year. Once this is done, each developer can post any number of apps on Google Play.

iOS

Apple, unlike Google, has a strict validation process, and has total control over the contents of its App Store. A new application posted to the store can take **1 to 3 weeks** to be validated, and the process takes 2 to 10 working days for a new update of an existing application.

Apple requires developers of iOS applications to use only the official Apple code libraries and SDK. Any use of an outside API causes the application to be rejected. Other factors for rejection are copyright infringement, discriminatory or pornographic content, etc.

Statistical and analytical monitoring

When the mobile website or application is in production, the resulting user monitoring and data analysis respond to 2 categories of needs:

- **Marketing:** analysis of user behavior (purchase funnel, etc.), segmentation, and so forth. Metrics such as the number of downloads, number of users, or the most frequently visited pages are essential data for understanding how your application is actually being used.
- **Technical and troubleshooting:** to identify the causes of slow service and any technical problem that users encounter during when using the site or application.

Mobile website analytics

For websites, solutions for monitoring user activity are technically implemented by JavaScripts and HTTP cookies to:

- reconstitute user journeys
- follow a user over time: new visitor or returning visitor?
- track business indicators such as the bounce rate

For AMP-optimized HTML, there are fewer possibilities. Website analytics must be based on the dedicated analytics libraries supplied by the AMP project.

Mobile app analytics

There are three main categories of analytics for mobile applications:

- **App Marketing Analytics:** these offer insights into the visibility of your application in stores, and trace the number of downloads and use of the app.
- **In-App Analytics:** these analyze how users behave when using your application, which pages they visit, etc.
- **Crash & Performance Analytics:** these analyze the availability of your application, detect crashes and categorize them by type of device, etc. To analyze slow performance and the internal workings of an application, the subject of ‘troubleshooting’ will be discussed below.

Among the best known analytics tools for mobile applications are Google Analytics (mobile websites and applications), Mixpanel, Flurry, and Fabric (formerly Crashlytics).

Some tools cover mobile websites and mobile apps on a single control console. These will be discussed in the following chapter.

Analytics tools

These take the form of a section of code (JavaScript, often via a tag management tool) embedded in a mobile website or application (SDK).

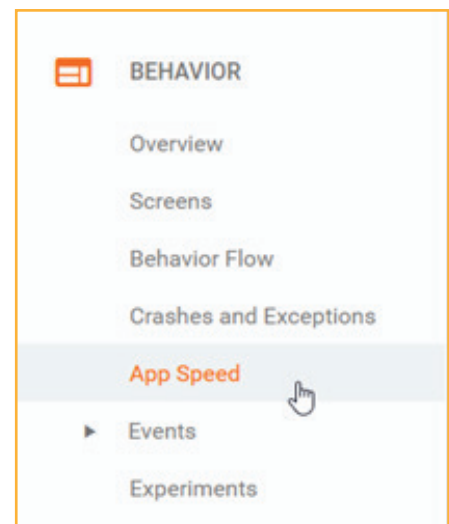
Google analytics (Mobile websites & apps)

Google Analytics (<https://www.google.com/analytics/>) is a free tool for basic functionalities. It provides information on user behaviors, mainly for the needs of marketing (conversion of visits into payment actions):

- **Acquisition /Audience:** distinguishes between new visitors and returning visitors, plus their country, language, and version of the application they are using.
- **Behavior/Engagement:** creates actions to track users and generate reports.
- **Conversion:** allows you to set targets, tracks conversion against targets, and analyzes results.

For mobile applications, troubleshooting features are available in the submenus 'Crashes and Exceptions' and 'App Speed'. The latter provides metrics which must be coded explicitly, and are therefore not automatically available:

Google's Firebase Performance Analytics works according to the same principles of dedicated instrumentation in the application.



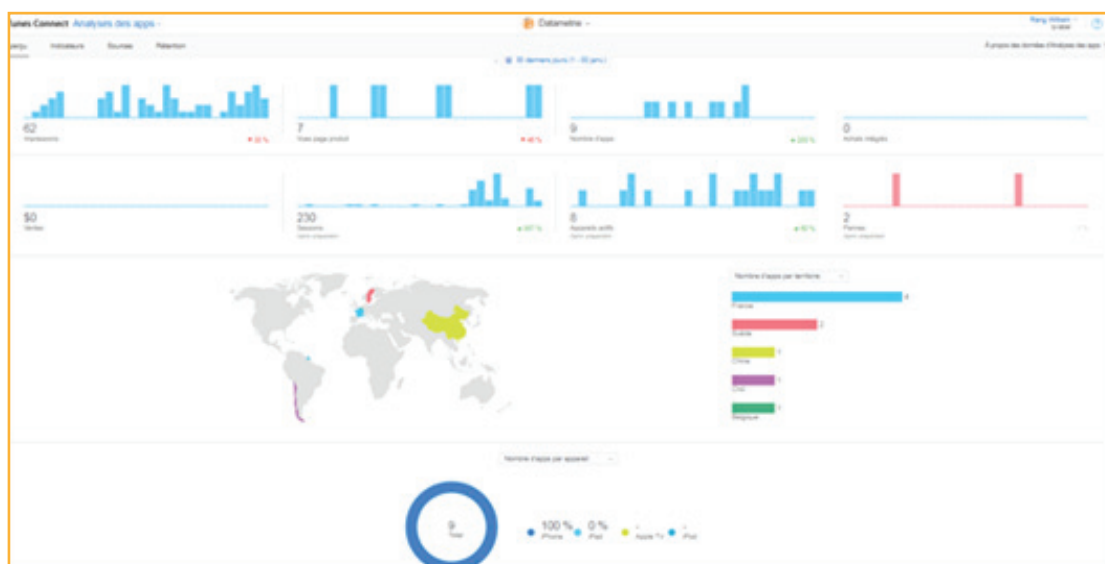
Apple App Analytics (iTunes Connect)

Apple's analytics solution is free, and offers 3 tools to analyze your data:

1. App Store data
2. Sales data
3. Usage data

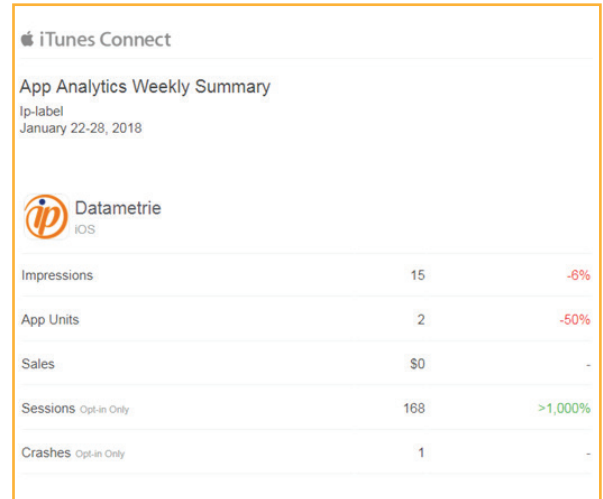
These tools from Apple trace the number of purchases, the number of installations, and the user of your application, all by geographical area. You can also obtain the number of times your application is viewed in the App Store and determine whether the users who installed it found it by browsing the store, by searching the store, or from a referring application or website.

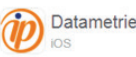
When a new iOS version is released, the developer can, for example, identify any drop in the number of downloads of the application if it is not compatible with the new iOS version.



Source: ip-label

Apple Analytics also offers a weekly e-mail report service containing the main metrics : number of impressions, installations, sales, sessions, and crashes (for each application).



App Analytics Weekly Summary		
ip-label January 22-28, 2018		
		
Impressions	15	-6%
App Units	2	-50%
Sales	\$0	-
Sessions <small>Opt-in Only</small>	168	>1,000%
Crashes <small>Opt-in Only</small>	1	-

Source: ip-label

Troubleshooting tools

In addition to tracing user behavior (with analytics), any malfunctions must also be identified and diagnosed:

- Crashes
- Application errors and API errors
- Any performance problems and bottlenecks (see Chapter 4)

As mentioned above, audience monitoring tools are a first approach to crash detection, but a dedicated solution is needed to drill down to details on application errors or calls to external APIs (third-party search services, notifications, etc.), which are more complex to capture and report.

Troubleshooting mobile applications

A troubleshooting tool is inserted into a mobile application as a third-party library (SDK) when it is compiled. When it runs on each user's device, it automatically captures errors in the application and external APIs that are called in the code.

Crashlytics monitors reports of the crashes that occur on the application. Information is available on the web interface, as well as on a dedicated application. Following its takeover by Fabric, the tool has been integrated into a larger development framework.

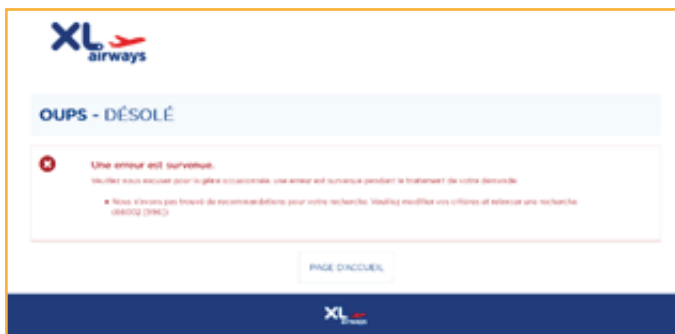
Another example of a performance-oriented SDK is **Firebase Performance Analytics**, which can capture response times by start/stop timers.

Troubleshooting web & Hybrid web apps

Web applications and hybrid mobile applications use HTML pages and JavaScript to display on user devices. In the case of hybrid applications, an embedded browser displays the page (WebView). Monitoring the performance of WebViews is also possible with an embedded JavaScript code using the W3C Navigation Timings API:

- DNS and TCP times (network)
- Server response time
- Load time

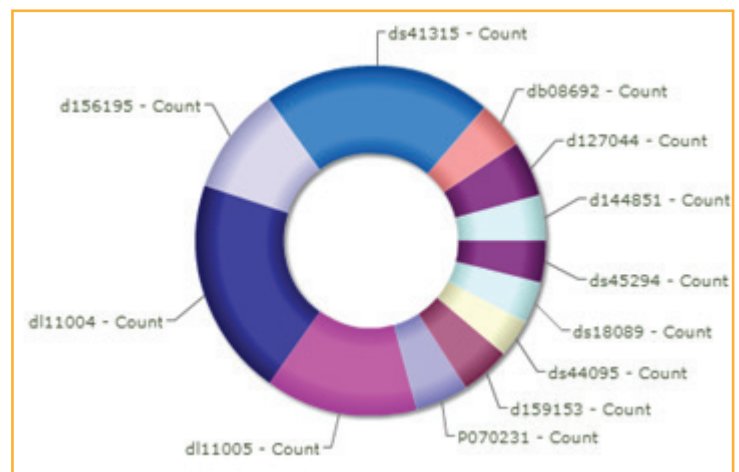
Monitoring of actions can be customized with the ip-label JavaScript (RUM BI) via start/stop timers.



To track errors on a mobile web page, a capture is made of standard messages generated by the application, particularly business error codes. In fact, such application-generated messages are integrated into the page's HTML code, which makes them exploitable by purpose-built tools.

This makes it possible to monitor trends and break down errors by type (or by business code) as shown in the graphic below (5-digit error code).

The interpretation of these errors depends on the typology defined in the business application.



Source: ip-label



N°5

Performance |

The matter of performance in mobile usages

Performance is an important part of the user's digital journey. In fact, recent studies like the one from Doubleclick show that a full loading of a page correlates strongly with marketing indicators for conversion. Over half (53%) of all users abandon a mobile website if the page doesn't load within 3 seconds (<https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/>).

The aim is to optimize performance in order to increase the success rate of marketing objectives, including the bounce rate or conversions.

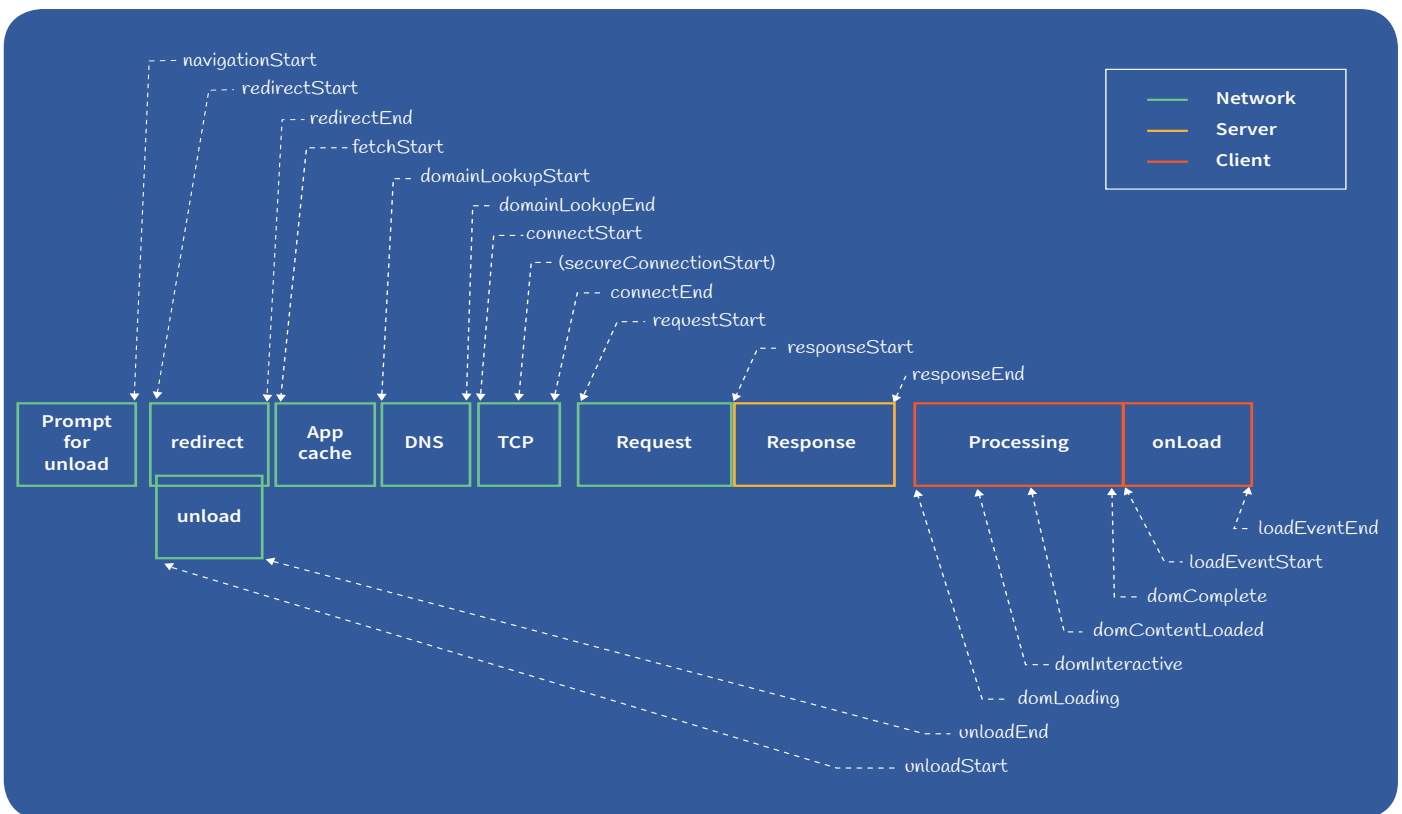
You need to monitor performance indicators to identify which users encounter technical problems and bottlenecks. Performance metrics adapt to the type of application: mobile websites and any kind of mobile app (iOS/Android).

The following chapters discuss the KPIs that formalize these objectives and keep track of the results of improvement actions over time.

Mobile website performance

For **websites (responsive or mobile-dedicated)**: we advise measuring the end of loading and the possibility of interacting with the page, using the metric '**Time To Interact**' or TTI, which has been standardized by the W3C and is available via the Navigation Timing API aboard recent

generation browsers. This metric allows you to determine the moment when the page is available for the user to interact with it (click, scroll, etc.). The TTI metric is named "domInteractive" in the diagram below from the W3C:



Source : <https://www.w3.org/TR/navigation-timing/>

Furthermore, **Speed Index** is gaining popularity because it lets you determine how fast visual rendering is completed, by measuring the progression of the page display. It calculates an overall score closely reflecting the user's real-life experience. The value obtained is an index intended for comparison with other web pages. Speed Index is therefore particularly useful for:

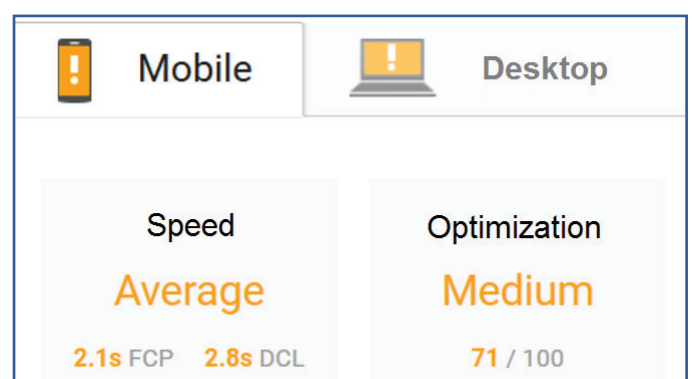
- comparing a site to competitor sites
- assessing the results of page optimizations during A/B testing

Find out more about the theoretical calculation of the Speed Index in this article: <https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index>

Free tools

The browser Chrome offers a 'DevTools' module which can simulate a mobile connection and a user agent or mobile screen size. This gives you a first general idea of the performance of a mobile website.

Google also proposes the **Page Speed Insights** tool which rates the desktop and mobile performances of a website:



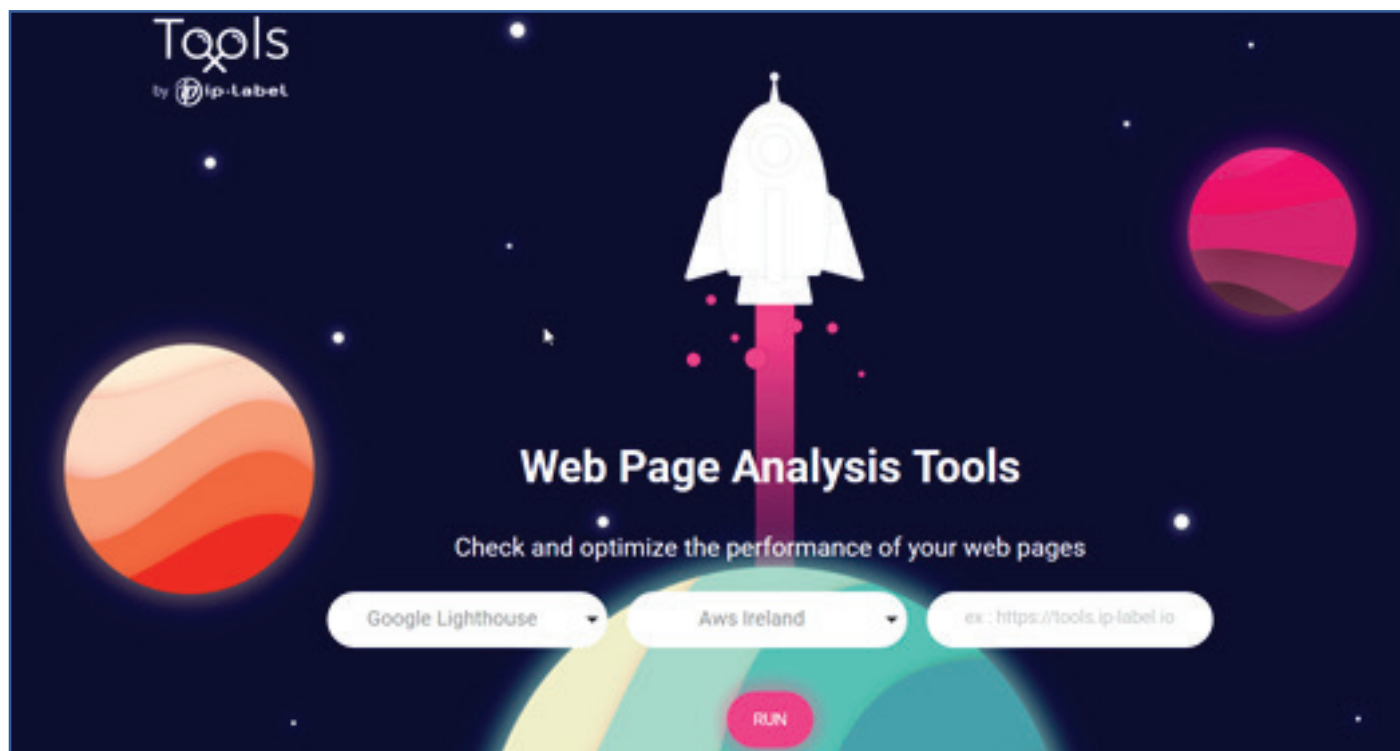
Source : <https://developers.google.com/speed/pagespeed/insights/>

As a reminder, the real-user metrics “FCP” (First content paint) and “DCL” (DOM content loaded) from the W3C Navigation Timing API on Chrome browsers have now been integrated into the score and results.

ip-label also offers automatic analysis tools (Google Lighthouse, etc.) from a calibrated, test-dedicated network core environment: <http://tools.ip-label.io>.

This site from ip-label provides site metrics and, in particular, calculates the **Speed Index** with Lighthouse.

Note that Lighthouse features in Chrome DevTools as well, for manual testing.

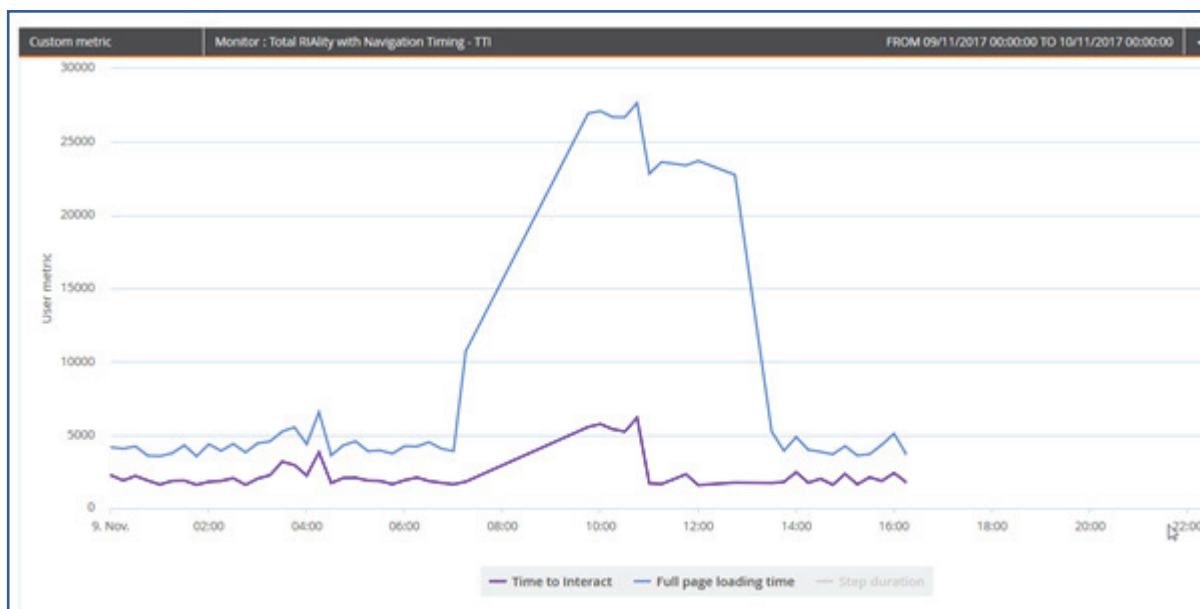


Automated measurement with robots

As manual testing by definition is occasional or ad hoc, and web pages are destined to evolve rapidly, it is essential to automate performance measurements in order to track them over time.

Robots can conduct measurements in a stable context to provide metrics of web page loading or mobile app screen loading (visual check).

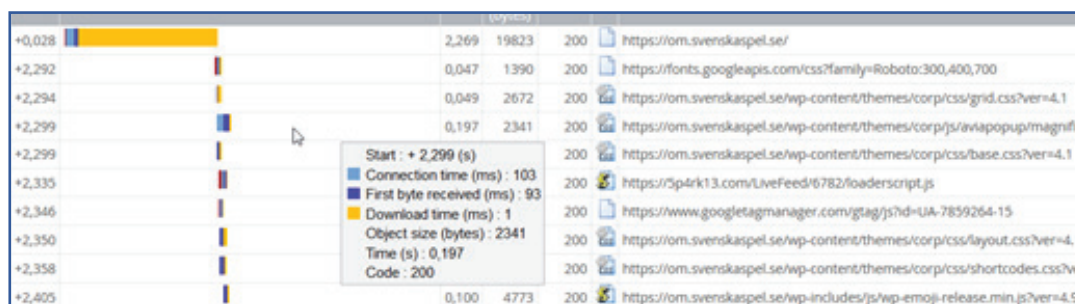
The example below shows the different **web page** times (Time to Interact, page load, etc.) measured by a robot on the internet network core (when necessary with a user agent emulation, mobile bandwidth, and screen size):



Source: ip-label

The TTI (Time To Interact) generally serves to better assess the visual impact of slowness for the user. The example above shows a disruption on external domains, which appears significant, but does not greatly affect the user's ability to interact with the page (disruptions on Google advertising domains).

Robot monitoring also allows detailed analysis of contents, as shown in the report below:



Source: ip-label

In addition to technical response times, good coding practices and performance indicators can be evaluated in the form of a score by such tools as Google Lighthouse or, very simply with the tool “Mobile friendly” (<https://search.google.com/test/mobile-friendly>).

Concerning **Speed Index**, the advantage of using a robot to measure performance is that any changes following updates, for instance, can be seen over time. Robots can also emulate mobile devices to measure the responsive version of a website.

Real-user web measurements (embedded code)

For web applications of mobile hybrid applications, it is possible to monitor real-user performance or customized actions with the ip-label JavaScript (RUM BI) via start/stop timers.

Monitoring the performance of browsers or WebViews with real users is therefore possible with an embedded JavaScript code using the W3C Navigation Timings API. The following metrics can be collected:

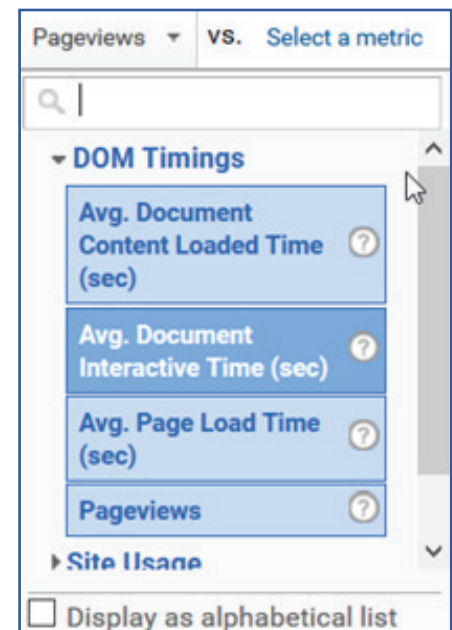
Web applications and hybrid applications use HTML pages and JavaScript to appear in a browser which supports W3C **Navigation Timings** performance measurement. In the case of hybrid applications, an embedded browser displays the page (WebView).

- DNS TCP time (network)
- Server response time
- Time to interact (TTI)
- Load time



For the **AMP format**, ip-label offers an AMP-dedicated RUM performance measurement library.

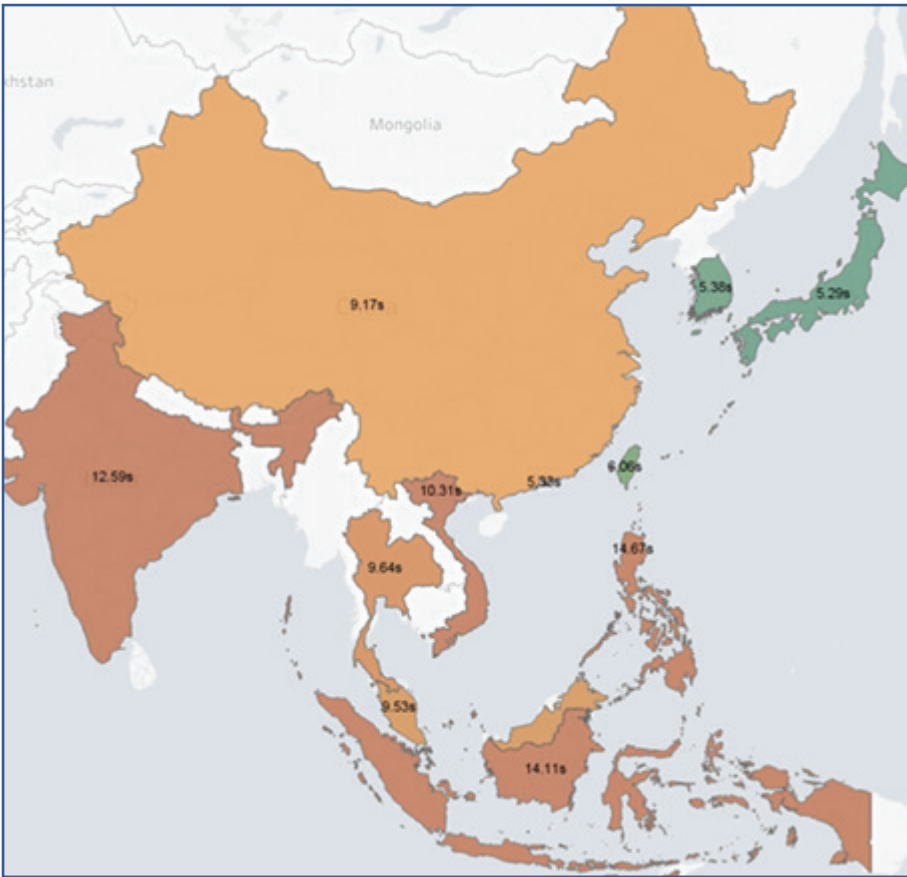
Analytics tools like Google Analytics can sometimes monitor ‘Time to interact’, under its ‘Site speed’ and ‘DOM timings’ menus:



Nevertheless it is important to be able to correlate performance with more complete business information. For example, ip-label provides the “custom dimension” feature to capture relevant information at the same time:

- Location (including in a private network)
- The user in question (ID)
- Version of the application
- Reference of the server used during the measurement
- Browser and device used: fixed or mobile, brand, model, OS version, etc.
- and more...

A data visualization solution is a vital asset for analyzing these dimensions case by case. The notion of geolocation (via IP address), for example, identifies the areas that are least well served, as the map below shows for Asia:



Source: ip-label mobile RUM (web)

Mobile application performance

For **native applications**, there is no technical standard for measuring the progression of visual loading on the screen. We advise differentiating between:

- the **time spent on the screen** by the user (“screen display” time in the ip-label SDK). This is a metric which closely reflects the user’s behavior, but also includes inactive time or the time it takes to read content.
- the **initial screen loading time**. As this is often made up of several modules (main class, WebViews, etc.), it is difficult to reconstitute. In this case a ‘custom’ metric can be set explicitly by the developer as a timer at meaningful points in the code.
- the **fluidity of refresh during use** can be measured in frames per second (FPS). The performance ideal is 60 FPS for an impression of maximum fluidity (for example when scrolling dropdown lists).

Free tools

Free performance measurement tools, like **Traceview**, are available in Android Studio: <https://developer.android.com/studio/profile/traceview.html>

This tool identifies the methods that use the most CPU, as shown below:

Name	Incl Cpu Tir	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Incl Real Time %
16 android.os.Parcel.writeInt (I)V	9.2%	0.562	5.5%	0.335	4.5%
Parents					
20 android.os.Parcel.writeValue (Ljava/lang...	53.4%	0.300			53.4%
9 android.os.BaseBundle.writeToParcelInn...	15.5%	0.087			15.4%
18 android.content.Intent.writeToParcel (L...	14.4%	0.081			14.5%
5 android.app.ActivityManagerProxy.start...	9.1%	0.051			9.0%
13 android.os.Parcel.writeArrayMapIntern...	4.8%	0.027			4.7%
119 android.net.Uri.writeToParcel (Landro...	2.8%	0.016			3.0%
Children					
self	59.6%	0.335			59.7%
33 android.os.Parcel.nativeWriteInt (JI)V	40.4%	0.227			40.3%
17 android.os.BaseBundle.putInt (Ljava/lang/Str...	8.1%	0.498	1.2%	0.072	4.2%
18 android.content.Intent.writeToParcel (Landro...	8.1%	0.496	0.5%	0.033	4.0%
19 android.util.ArrayMap.put (Ljava/lang/Object...	7.3%	0.447	3.4%	0.210	3.8%
20 android.os.Parcel.writeValue (Ljava/lang/Obj...	7.1%	0.437	1.6%	0.098	3.5%
21 android.view.IWindow\$Stub.onTransact (ILan...	6.9%	0.420	1.1%	0.065	5.4%
22 android.app.Activity.isTopOfTask (I)Z	6.7%	0.410	0.2%	0.012	5.0%

Also from Android, the ‘profiling’ tool, **Hierarchy Viewer** measures how long it takes for views to display (Draw time): <https://developer.android.com/studio/profile/hierarchy-viewer.html>

For iOS, the equivalent profiling tool in Xcode is “**Instruments**”.

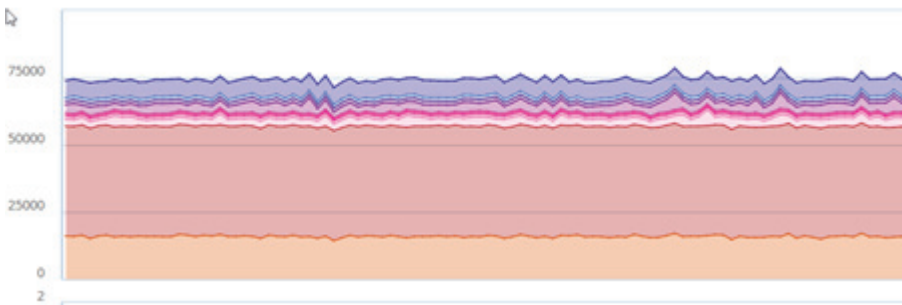
These tools nevertheless require advanced expertise to yield meaningful metrics.

Automated measurement of mobile applications

Automated measurement is performed by a robot to return the time a test user spends on the screen, with visual checks that can be broken down into the steps of a transaction.

The advantage of measurement robots is that they are independent of the framework used to develop the mobile application.

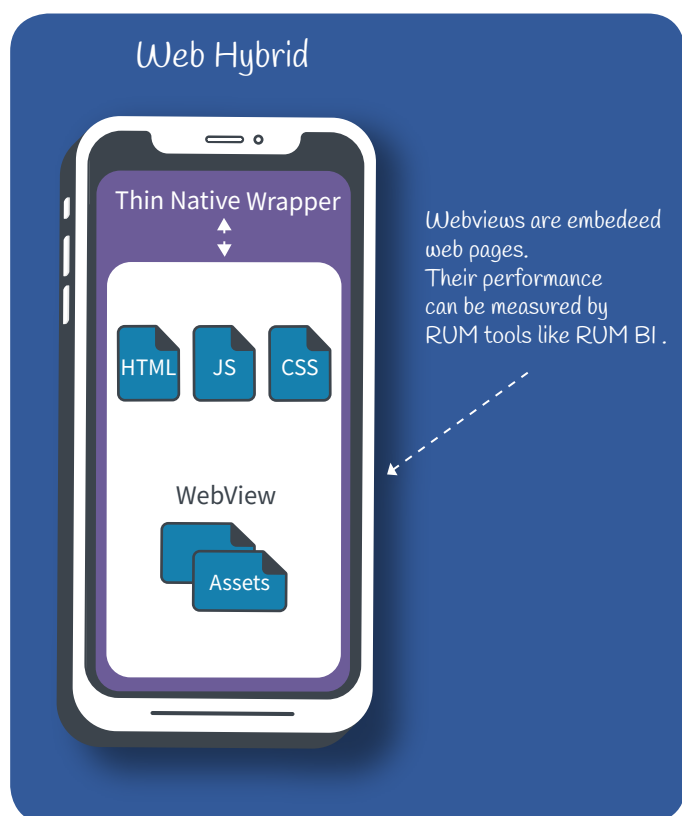
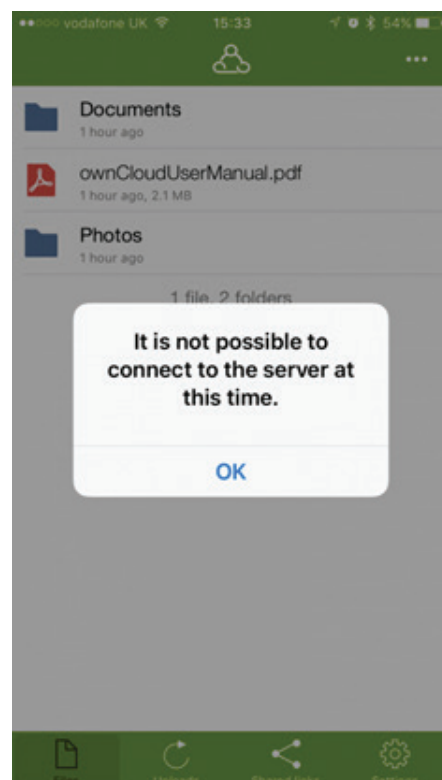
The example below shows a transaction monitored around the clock on an iPhone mobile application, with very little variation over time:



In the event of a disruption, a screenshot can identify the step of the transaction where the stall occurred, as illustrated below:

Another characteristic of robot measurement is that it can perform visual checks of the display, and therefore is able to take into consideration any technical change included in the screen (WebViews, etc.) to check **transaction integrity**.

Furthermore, native applications often include ‘WebViews’, embedded “mini web browsers” which can return web-oriented metrics from real users (RUM), including TTI (Time To Interact).



Note that in this case, the TTI metric is available via the WebVis of the screen which also supports the Navigation Timings standards from the web (W3C) via the JavaScript API. Using the measurements

made with JavaScript is also useful for **hybrid applications** that integrate web pages, like Cordova.

The Android virtual mobile robot

For Android, a virtual terminal can be deployed in the cloud to virtualize installed applications.

A robot monitoring solution can therefore be 100% virtualized as illustrated below:

This solution makes it possible to easily deploy Android configuration variants in accordance with specific needs, for example for regional markets (Europe, USA, etc.), or different versions (Android 7/8/9/..).



Active + passive monitoring: a winning strategy for mobile performance

At present, any mobile application performance management solution must include the following two important approaches:

1. Monitoring of **service availability 24 hours a day**, using test robots which simulate user transactions. When service is slow, the robots stable context makes it possible to substantiate the real performance experienced by users (visual display or Speed Index).

Real devices can also be used to run tests on iOS or Android. For Android, a *cloud virtual device* can be planned, to reduce the costs of deployment.

2. **Real-user monitoring (RUM)** via an embedded code (Javascript or SDK) dedicated to the collection of performance metrics.

Reflections on mobility, performance, and opportunities

A strong trend toward all things mobile is evident everywhere. In some domains, such as e-commerce, developed countries have already seen a higher percentage of transactions on mobile applications than on desktop web.

Beyond technical aspects, the choice between designing a single responsive web site across desktop/mobiles/tablets and designing web sites (or applications) dedicated to each environment must also consider the complexity of the products.

A complex product probably requires a dedicated purchase funnel for each environment to optimize the purchasing process. The notion of “mobile first”, which aims to offer the user a real mobile-dedicated experience, takes on a whole new meaning.

Whatever the technical choice, for each environment we have seen that monitoring solutions exist to correlate business indicators with technical metrics.

Furthermore, mobility is one aspect of new technologies where progress is shared everywhere in the world. In a lot of fields, the ‘digital divide’ continues to split the world into developed countries on the one hand, and emerging economies on the other. With mobile, this is much less so. In China, for example, most people do not have a computer at home, but everything mobile is very popular and widespread.

This is due to the fact that mobility requires less costly infrastructure than other ICTs, like fixed telephony or desktop computing. For this reason, instead of trying to catch up with fixed infrastructures which are expensive to set up, many developing countries were quick to move to mobile.

Mobility is therefore a domain relatively immune to ‘digital divide’ symptoms, and it therefore offers very interesting opportunities for development. This should further motivate innovations around a greater variety of services, to reflect the diversity of users around the world.

We hope this white paper contributes to helping marketing and development teams to see eye to eye on the importance of mobile performance metrics, with a view to significantly improving application performance and reaping business benefits.

About the ip-label group

ip-label has been assisting enterprises in their digital transformation for over fifteen years. A leading specialist of user quality of experience, the ip-label group offers an extensive range of APM (application performance management) solutions for analyzing and measuring the performance of all digital services including web, business apps, mobile, telephony, video and voice.

Want to know more about application
performance management?

[Request a demo](#)



www.ip-label.com

+33(0)1 77 49 53 00

marketing@ip-label.com



Paris | Madrid | Stockholm | Helsinki | Shanghai | Tunis

©ip-label 2019. All rights reserved