# Comment garantir votre performance mobile ?

Stratégie de conception, critères de performance et expérience utilisateur...


ekara by ip-label

# Sommaire

# Introduction

## Le mobile : un enjeu majeur de la transformation digitale

La multiplication des appareils et usages mobiles (smartphones, tablettes, laptops…) a profondément **restructuré le marché des nouvelles technologies**. Le mobile est un volet essentiel de la transformation digitale de toute entreprise et de tout organisme, quelle que soit son activité principale (commerce, logistique, éducation, transports, santé…).

Ces dernières années ont été marquées par une forte augmentation des achats sur mobile, particulièrement en Europe. Le mobile s'impose aujourd'hui comme le terminal privilégie devant l'ordinateur. Selon la FEVAD, l**e mobile représente les 2/3 des audiences des sites e-commerce, et près de 50% des transactions sont réalisées désormais sur mobile.**

Par ailleurs, **le nombre de recherches Google effectuées à partir de terminaux mobiles a dépassé le nombre de recherches faites sur les PC.** Google prend en compte la performance mobile pour ses résultats de recherche depuis 2018, ce qui implique de prendre en compte les indicateurs de performance le plus tôt possible dans le développement ainsi qu'en production.

Pourtant, **mener à bien un projet mobile peut s'avérer délicat si l'on ne prend pas en compte tous les leviers de succès.**

Si les performances du service mobile sont défectueuses, et par conséquent le ressenti utilisateur défavorable, cela peut nuire fortement à l'image de marque, à la productivité, voire provoquer une baisse du chiffre d'affaires. L'**expérience utilisateur est déterminante** pour la réussite d'un projet mobile.

Comment se préparer ? Quelles sont les stratégies possibles ? En quoi la performance est-elle fondamentale pour réussir un projet mobile ? Découvrez les bonnes pratiques dans ce livre blanc.


N°1

# Design

Quelle stratégie
de conception adopter ?

# Quelle stratégie de conception adopter ?

## Définir son projet mobile

Avant de démarrer tout projet, il convient de définir précisément les objectifs, la cible du projet, puis les besoins en termes de compétences techniques.

Ces objectifs peuvent être, par ordre de priorité ou de maturité :

1. **De développer une présence web mobile performante** sur tous types de terminaux, pour attirer de nouveaux utilisateurs/clients. Par exemple via les réseaux sociaux, les moteurs de recherche ou les campagnes publicitaires. En effet, la croissance du web mobile est forte et inexorable d'année en année

2. **De fidéliser ses utilisateurs**. Ici, une étape importante sera le lancement d'une application mobile, pertinente suivant le cœur de métier.

3. **De définir un nouveau business model** : abonnement numérique via l'App store, achat intégré dans l'application, etc.

Dans tous les cas, les besoins en compétences techniques sont également cruciaux : développeurs Web, développeurs avec des compétences mobiles dédiées iOS/Android ou expertise sur les frameworks multi-plateformes comme Flutter ou React Native.

**Le choix de s'investir** dans un site web adapté aux usages mobiles, dans une application mobile, ou dans les deux à la fois se décide en fonction des objectifs du projet, des compétences et ressources humaines disponibles, et du budget pouvant être mobilisé.

Site web mobile ou application mobile, quels sont les avantages et inconvénients ? Voici ci-après quelques éléments de réponse.

# Un site web mobile dédié ou responsive

Selon les besoins de l'entreprise et de ses clients internes et externes, les sites web mobiles peuvent être soit le point d'accès mobile unique, soit une ressource complémentaire aux applications mobiles.

Le premier choix de design web mobile se fait entre un site web dédié aux utilisateurs mobiles ou un site web "responsive" :

- **Dédié :** on pense ici au mobile comme cible n°1 avec des templates et des ressources js/css dédiées, de même que le contenu avec des ressources adaptées comme les images.

- **Responsive :** le site web utilise le même socle de base pour les ressources (notamment CSS, mais aussi javascript) pour tous les utilisateurs Desktop et mobile.

## Le site web dédié au mobile

**Le site web mobile est un site web dédié à une utilisation sur tablette ou smartphone.** Le contenu et les fonctionnalités de ce type de site web sont optimisés pour prendre en compte les spécificités des terminaux mobiles.

Les pages sont spécialement conçues pour le confort d'utilisation sur appareil mobile, entre autres par sa présentation à l'écran, les fonctionnalités tactiles, les menus, etc.

### ⊕ Les plus

Le site web dédié mobile offre une rapidité d'affichage et un confort de consultation sur appareils mobiles. Techniquement, il s'agit de l'une des solutions les plus simples pour se doter d'une présence web mobile.

### ⊖ Les moins

Il n'y a pas de compatibilité avec les sites pour ordinateur (URL différents) ; il faut développer et gérer les sites web mobile et fixe séparément afin d'assurer une présence web adaptée à tous les terminaux. Ceci implique des coûts accrus de création, de maintenance, et de mises à jour du site web mobile.

En termes de **performance**, les principaux critères à prendre en compte ne sont pas très différents de ceux qui améliorent la performance de tout site web : le volume total des ressources chargées sur le réseau et leur ordre de chargement, la compression, le nombre de requêtes HTTP, et l'allègement du traitement côté client pour l'exécution de JavaScript et les widgets tiers.

Pour résumer, le design du site doit permettre un affichage rapide de toutes ses pages et une bonne performance de tous ses services pour les utilisateurs en situation de mobilité.

Nous verrons plus bas dans le chapitre sur les frameworks une variante de site web dédié avec l'initiative AMP créée par Google.

## Le site web adaptatif ou "responsive design"

**Le site web 'responsive' est un seul et même site qui s'affiche sur tous les types de terminaux** (tablette, téléphone, moniteur d'ordinateur).

C'est un site qui s'adapte automatiquement à la taille d'écran du terminal utilisé. Cette adaptation automatique de l'affichage utilise les informations fournies par le terminal lors de la création ou mise en style de la page ('media queries' en CSS).

Ceci a pour conséquence d'augmenter les options d'affichage à gérer par les développeurs, et doit être pris en compte dès le début. Il est possible, par exemple, d'afficher ou de cacher certains éléments, de réduire la hauteur des menus, de transformer les colonnes en lignes pour un affichage vertical, etc.

Bien que la gestion d'un « layout » différent par taille d'écran complexifie la conception du site, les frameworks du marché (ou templates des gestionnaires de contenu) facilitent le travail.

### ⊕ Les plus

Les coûts sont maîtrisés car il s'agit d'un seul et même site avec une administration unique. Les mêmes développeurs avec compétences HTML/CSS/JavaScript peuvent adapter le code existant pour une restitution mobile.

### Les moins

L'affichage s'adapte à l'écran mais le contenu (vidéos, fonctionnalités ou menus) n'est pas forcément optimisé en situation de mobilité. Ceci peut poser un problème de poids sur le réseau et de ressources utiles au rendu (CPU principalement). On peut néanmoins adapter les assets en fonction du terminal : ne pas charger certaines portions du site comme les backgrounds ou les carrousels d'images par exemple...

De plus, il faut tester chaque page sur les différents appareils pour vérifier qu'ils affichent correctement les contenus, alors le temps de développement est plus long.

# Les applications mobiles

## L'approche native

On parle de natif lorsque l'application est développée avec les outils et langages proposés par le système d'exploitation propre à une plateforme, par exemple Java pour les plateformes Android, Objective-C et Swift pour les plateformes iOS.

### Les plus

Le natif est un gage de performance car les chargements sont plus rapides parce qu'ils se font localement. De plus, l'application est accessible hors connexion. Le développement natif permet facilement d'adapter son application en fonction du type de plateforme (terminaux iOS ou Android). On note également un meilleur référencement grâce aux plateformes de téléchargement.
En résumé, les avantages sont :

- La performance
- L'UX maitrisé
- Les notifications push
- Le mode Full screen

### Les moins

Lorsque l'on souhaite développer une application native sur plusieurs plateformes, il faut maintenir autant de codes sources qu'il y a de plateformes cibles. Chaque plateforme ayant ses propres spécificités, ceci alourdit les coûts de développement et des mises à jour.

Développer dans l'environnement natif Android ou iOS requiert des compétences techniques spécifiques. Il est néanmoins possible, pour gagner du temps, de s'appuyer sur des frameworks pour déléguer certaines fonctionnalités comme l'hébergement ou la base de données.

Le coût financier est plus important si l'on souhaite développer une application pour les terminaux Android et iOS à la fois, dans un langage de programmation propre à chacune des plateformes.

Pour conclure sur la tendance la plus récente, d'après la dernière étude et l'index TIOBE (voir https://www.tiobe.com/tiobe-index/), l'adoption de SWIFT et d'Objective-C est en baisse, au profit du framework Flutter qui simplifie le quotidien des développeurs en ne créant qu'un seul code pour toutes les plateformes : Web/Android/iOS.

## Les frameworks multi-plateformes

On parle de multi-plateformes lorsque le framework génère un seul code pour toutes les plateformes : web, desktop, natif iOS et Android.

Cela permet de compiler une application pour toute plateforme.

### Les plus

Le temps de développement pour une application est fortement diminué car le code source est commun pour chacune des plateformes cibles. La maintenance aussi est plus simple du fait que les révisions sont faites sur une seule version.

### Les moins

Cette approche est jeune et peut donc être instable.

La question de la perennité de ces frameworks se pose également, Google et Facebook abandonnant souvent des projets en cours.

Autre point négatif : les performances ne dépassent pas celles d'une application développée directement en langage natif.

## Les Progressive Web Apps

Les « progressive web apps » ou PWA se basent sur des pages web de type SPA (Single Page App) pour offrir une expérience utilisateur proche de celle d'une application mobile native.
https://developers.google.com/web/progressive-web-apps/

Les caractéristiques clés sont :

- Icône sur l'écran d'accueil (favori)
- Lancement en plein écran (Immersive full-screen experience)
- Notifications "push"
- Chargement immédiat, indépendamment de l'état du réseau
- Rapide et fluide : ressources pensées dès le départ pour le mobile

Les prérequis techniques :

- Des composants d'UI conçus avec ces contraintes,
- La déclaration du manifeste
- L'usage des service workers (pour le cache local et le fonctionnement hors-ligne) et du stockage local

Parmi leurs caractéristiques 'app native', les PWA permettent notamment d'utiliser les capacités du téléphone (notifications, micro, appareil photo, GPS), et se chargent immédiatement, indépendamment de l'état du réseau (fonctionnement hors connexion) en s'appuyant sur un proxy local ('service workers') instrumenté par le développeur.

Quant aux caractéristiques 'web', les PWA adaptent leur affichage sur tout terminal, mobile ou fixe. De plus, comme ils se basent sur les technologies web (HTML, JavaScript); aucun fichier compilé natif (APK ou IPA) n'est produit. Les PWA ne dépendent d'aucune plateforme de téléchargement pour leur diffusion et - comme pour visionner un site web – on accède très rapidement à la phase d'interaction. Le référencement par les moteurs de recherche se réalise comme pour tout site web et permet donc d'utiliser le même SEO.

### ⊕ Les plus

Les avantages des PWA sont les mêmes que ceux des applications mobiles natives ajoutés à ceux des sites web mobiles.

Les plus :

- Les notifications push
- Offline
- Icone
- Full screen
- Instant loading

Tous les frameworks web utilisés actuellement (React, Vue, Angular) génèrent nativement une PWA lors de la création d'un projet d'application web.

### ⊖ Les moins

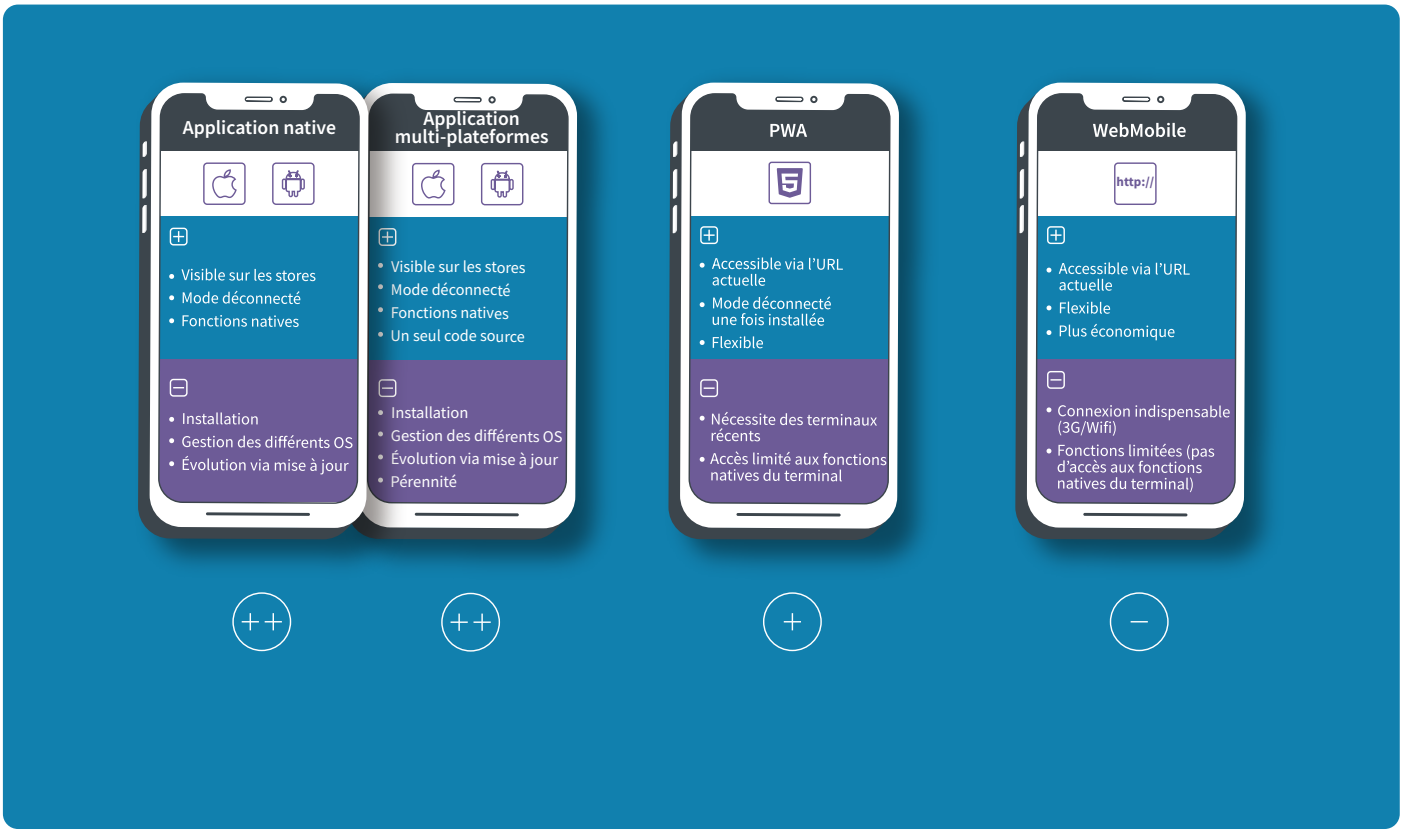
Le principal inconvénient des Progressive Web Apps est qu'elles ne sont pas compatibles avec l'App Store. Puisqu'il est impossible de publier une PWA sur l'App Store, il est donc impossible de cibler les utilisateurs Apple.

La publication sur Google Play est également plus difficile en comparaison avec une app native.

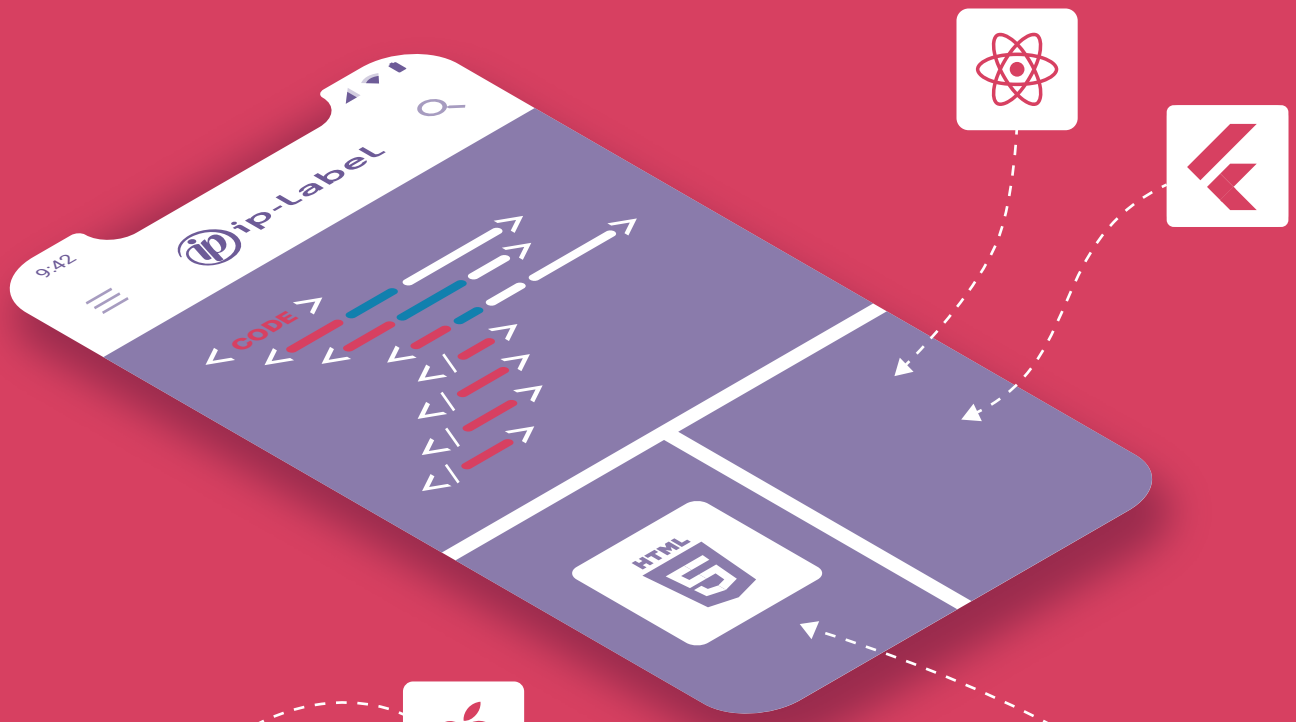
Autre bémol, les notifications push ne sont pas possibles aujourd'hui avec les Progressive Web Apps.

# Synthèse des choix de conception

Les avantages et inconvénients des choix de conception sont résumés ci-dessous :



Frameworks




Native app
vs
Cross-platform app

N°2 **Développement**

# Le développement d'une application mobile

## Le choix de l'environnement (IDE)

La première étape dans la phase de développement mobile consiste **à choisir la plateforme sur laquelle vous souhaitez déployer votre produit ou application mobile.**

La quasi-totalité des applications natives est développée sur Android de Google ou sur iOS d'Apple. Les développeurs doivent néanmoins faire face à un nombre accru d'environnements de développement (les « IDE » ou « integrated development environments »). **Le choix d'un environnement de développement 'officiel' ou 'adopté' par les éditeurs de systèmes d'exploitation mobiles permet d'avoir accès à toutes les fonctionnalités fournies par ces derniers,** des mises à jour sur le produit, et une aide et une communauté de développeurs assez riche.

Il est également possible de créer une seule et même application en utilisant un framework multi-platformes.

### Xcode - iOS

*Langages : Objective-C / Swift*

Xcode est l'environnement de développement natif et officiel d'Apple pour le système d'exploitation mobile iOS. Il est fourni avec une suite logicielle complète pour développeurs. Le développement sous XCode se fait essentiellement à travers les langages Objective-C ou Swift (disponible uniquement sur Mac OS).

## Android Studio - Android

*Langages : Java / Kotlin / Dart / TypeScript*

Google a fourni à son tour son IDE ou environnement de développement officiel et supporté « Android Studio » pour le système d'exploitation mobile Android. Cet environnement remplace Eclipse depuis 2014. Le développement se fait sur cet environnement à travers le langage Java essentiellement. En 2017, Google a adopté le langage « Kotlin » créé par la société « JetBrains » comme étant le deuxième langage de développement pour son OS Android.

## Netbeans - Android

*Langages : Java / Kotlin / Dart / TypeScript*

À l'instar d'Eclipse, Netbeans est un IDE utilisant le Java. Un pack « Netbeans Mobile » est disponible, donnant accès à une suite logicielle complète pour pouvoir développer sous Android. Netbeans est un environnement qui donne accès, selon la version, à faire du développement sur différents types de technologies et de plateformes comme le développement web ou mobile. Comme pour Eclipse, les développeurs auront besoin du plugin pour Android.

## VS Code – Android

*Langages : Java / Kotlin / Dart / TypeScript*

Développé par Microsoft, c'est devenu l'éditeur le plus populaire chez les développeurs qui l'utilisent déjà pour des projets autres que mobiles.

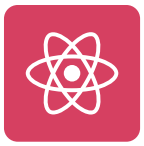
Les développeurs utilisant des frameworks multi-platformes peuvent préférer utiliser VS Code pour sa souplesse et sa base de plugins existants.

# Le choix du framework

Un framework est un ensemble de briques ou composants logiciels réutilisables, à la base d'une application ou d'un logiciel.

Il existe des centaines de frameworks couvrant une majorité des langages de programmation. L'objectif d'un framework est de simplifier le travail des développeurs. Les principaux avantages d'un framework sont la **réutilisation de code** et la **standardisation** du développement du logiciel.

## Frameworks multi-plateformes



### React Native

React Native est la solution de développement multi-platformes développée par Facebook. Elle a l'avantage de pouvoir capitaliser sur des compétences de développement Web largement répandues (javascript, typescript) tout en permettant de générer des applications natives (IPA ou APK).

React Native est apparue en 2015, et fait preuve d'une large palette de fonctionnalités et d'une stabilité éprouvée. Ce framework bénéficie d'une grande communauté d'utilisateurs.

### Flutter

Flutter peut être vu comme la réponse stratégique de Google à Facebook pour le développement d'applications mobiles, mais il est, en revanche, basé sur un langage de développement très peu utilisé jusque-là, Dart, et non du Javacript comme React Native. Flutter se base sur l'utilisation graphique d'un 'canvas' où les objets de l'application peuvent être dessinés.

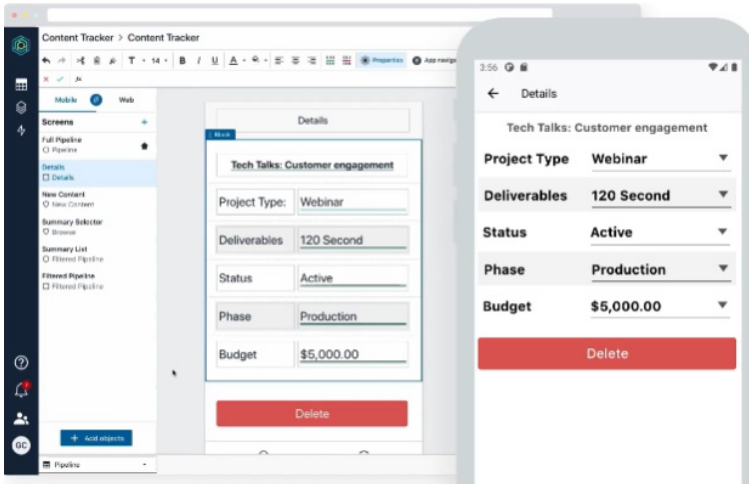
## Les environnements de développement no code

Il existe aussi des solutions pour développer rapidement avec une interface visuelle (drag and drop) des applications simples.

On peut citer notamment parmi les plus connues :

1. Amazon HoneyCode
2. Bubble.io
3. AppSheet

Ces solutions intègrent des thèmes prédéfinis. Tout l'environnement de développement est fourni et accessible dans le navigateur. Un exemple d'environnement de développement disponible est montré ci-contre :




*Source : HoneyCode*

## Frameworks de services cloud

### Firebase

Il est également important de mentionner Firebase qui est un framework particulier incluant à la fois des composants "client" (services pour application mobile comme le messaging, publicités...) et "backend" (Cloud Google) comme des bases de données ou de l'hébergement de librairies sur un CDN.

Firebase peut être utilisé à la fois via un langage natif (via un .jar dans Java Android par exemple, ou dans Objective-C ou Swift pour une application IOS par exemple) ou intégré dans n'importe quel environnement de développement mobile.

En plus des briques de développement, Firebase propose des fonctionnalités complémentaires de réseau publicitaire, de suivi du comportement des utilisateurs (analytics) et de suivi de performance, etc...


N°3

# Tests et recettes

# Les tests et recettes

L'évolution de la complexité des développements d'applications mobiles à travers le temps rend la réalisation d'une application avec zéro anomalie ou bug quasi impossible. La qualité des tests et de la validation faite sur les applications mobiles aura néanmoins un effet direct sur le succès de ces dernières auprès des utilisateurs.

Les tests peuvent généralement varier entre trois types :

1. **Tests unitaires** sur une fonctionnalité du code par les développeurs
2. **Tests fonctionnels** sur un enchainement de fonctionnalités, en tenant compte des différents contextes (taille d'écran...). Les **tests de non-régression** lors d'une mise à jour appartiennent à cette catégorie.
3. **Ergonomie et UX** : une fonctionnalité peut marcher, mais être difficile à appréhender pour l'utilisateur.

Le développement mobile complexifie les scénarios de tests en raison des facteurs suivants :

1. Taille variable des écrans : par exemple 1920*1080, 1368*768, etc
2. Mode "portrait" ou "paysage" pour l'orientation des pages
3. Modèle du terminal (et du navigateur pour une application web) : iOS, Android...

## Les tests unitaires

**Les tests unitaires sont des tests faits par le développeur pour vérifier le bon comportement d'une partie de son code ou d'une portion de son logiciel ou application développée.** La partie ou la portion vérifiée lors d'un test unitaire est appelée "unité" ou "module".

Les tests unitaires sont faits pour valider une fonctionnalité par rapport au résultat de retour attendu. Malgré leur importance, les tests unitaires ont toujours été une tache secondaire pour les développeurs. Il est à noter que pour ce faire la méthode XP « Extreme programming » insiste sur le principe des tests unitaires et les inclue dans le cycle de développement.

Le framework JUnit est un des frameworks les plus connus pour faire des tests unitaires en programmation et ce en langage Java. Il permet à travers l'utilisation du principe de l'AOP « Programmation Orientée Aspect » de définir les tests unitaires qu'on cherche à réaliser. La fonction « assert » permet de vérifier le résultat attendu lors d'un test unitaire.

Ces tests permettent d'aider le développeur à trouver plus rapidement les bugs et facilitent la maintenance du code source avec la documentation de celui-ci.

## Les tests fonctionnels (validation)

**Les tests fonctionnels sont destinés à tester la conformité du fonctionnement de l'application au résultat attendu, dans de multiples situations** (Terminaux différents, versions OS, taille d'écran, etc.). Ils sont impératifs pour garantir un niveau de qualité suffisant aux utilisateurs. Dans la pratique, les tests fonctionnels sont souvent effectués par **un panel d'utilisateurs**. Il existe des solutions de crowdsourcing pour atteindre une couverture de testeurs encore plus large (voir par exemple https://www.stardust-testing.com).

Au-delà de quelques testeurs, il est nécessaire de disposer d'un **outil de distribution privée** de type App store interne pour contrôler la diffusion à un groupe de testeurs, souvent faite pour de nombreuses mises à jour.

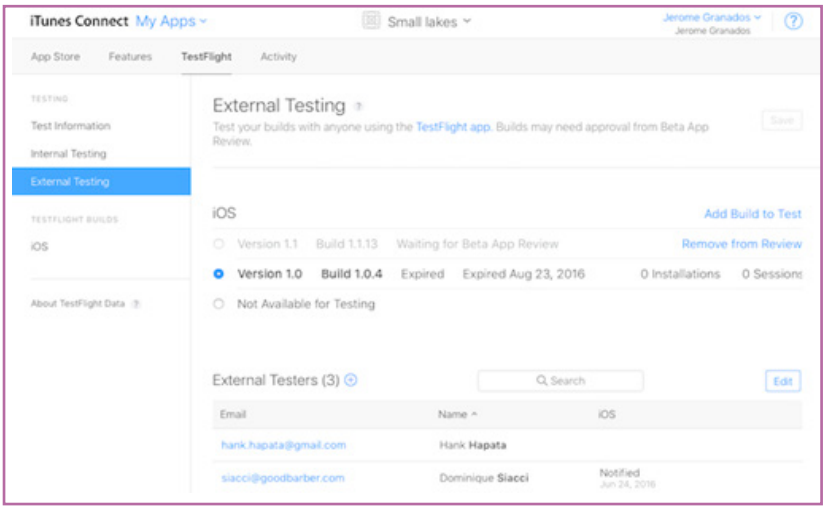
Pour Android, la diffusion n'est pas limitée et n'importe quel utilisateur disposant d'une version compilée (.APK) peut l'installer sur son téléphone, moyennant une notification d'Android mettant en garde contre la provenance d'une application disponible hors App store.

Pour IOS, il est nécessaire de connecter un terminal de test à Xcode ou de signer l'application avec l'ID d'un terminal souhaité, ce qui est très contraignant.

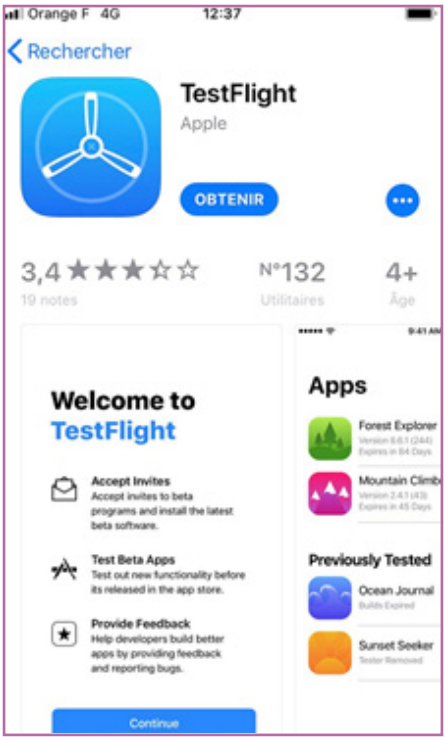
Certains outils facilitent néanmoins ce processus :

**TestFlight**

TestFlight est un outil gratuit (inclus dans le compte Apple developpeur), racheté par Apple, et qui permet de tester une application iOS avant sa mise en ligne sur l'App Store en la diffusant à un groupe de testeurs sélectionnés.
Vous devez utiliser le fichier compilé (.ipa) et le télécharger dans iTunes Connect, exactement comme si vous soumettiez votre application sur l'App Store.




Chaque utilisateur "testeur" doit installer de son côté l'application TestFlight depuis le store :




Il peut ensuite choisir les applications à installer depuis TestFlight, à partir d'une liste gérée par les développeurs.

Les testeurs sont de 2 types :

- Les tests en interne : Les testeurs sont ici des utilisateurs de votre compte iTunes Connect. Des personnes de votre équipe ou des personnes avec lesquelles vous avez partagé l'accès à votre compte. L'utilisateur devra saisir un 'redeem code' lorsqu'une application (build) lui est associée.
- Les tests externalisés : Pour un panel plus large ou externe à l'entreprise, vous pouvez inviter jusqu'à 2000 testeurs par projet sans qu'ils soient enregistrés dans iTunes Connect.
Attention : lorsque vous envoyez une invitation externe, Apple passera aussi en revue votre app pour approuver son test, comme pour l'app store normal.

Par ailleurs, Testflight permet également de récupérer les informations sur les crashs et offre des raccourcis pour capturer les remontées des "testeurs".

Il existe plusieurs alternatives à TestFlight, les plus connues étant **HockeyApp** (Microsoft) ou **Crashlytics Beta**.

**Automatisation des tests**

Au-delà de l'aspect fonctionnel, il est important de tester la robustesse et la stabilité de l'application en fonctionnement permanent sur une période suffisamment longue. L'automatisation des tests devient alors nécessaire. Pour l'implémentation de tests automatisés, on peut citer la librairie open-source Appium, intégrable dans des scripts Java, Python, etc. Dans un projet d'intégration continue mobile, ces scripts sont synchronisés avec la compilation via des outils comme Jenkins.

Il existe également des suites logicielles propriétaires qui permettent de réaliser des tests d'applications mobiles.
Ekara Mobile, par exemple, est la solution de mesure mobile proposée par ip-label. Cette solution permet de réaliser des tests sur de vrais terminaux iOS et Android. Les tests sont disponibles à partir d'une vingtaine de datacenters partout dans le monde. Ils vous permettent donc de réaliser les contrôles de temps de réponse à la fois en laboratoire sur vos environnements de recette mais aussi directement sur des environnements de production (voir plus loin).

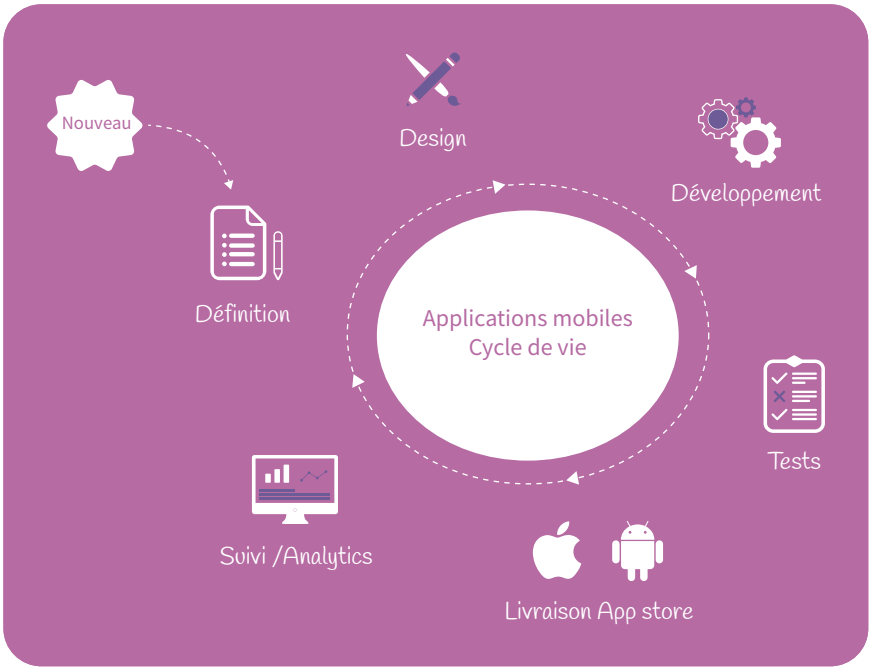
# Les tests d'intégration

**Le test d'intégration a lieu lorsque les différentes briques sont mises bout à bout pour :**

- Le backend/API
- Le frontend
- L'application mobile

La plateforme d'intégration utilisée sera différente de la plateforme de développement et proche de la plateforme de production.

Lorsque la chaine de production intègre à la fois la compilation et le déploiement de l'application avec des scripts de tests automatisés, on parle d'**intégration continue**. Le bénéfice est de contrôler rapidement la qualité de tout changement apporté dans l'application (code source).



Des outils permettent la mise en œuvre de l'intégration continue (Jenkins, Bamboo…) pour ces tâches critiques :

- Automatisation des compilations
- Automatisation des déploiements sur l'infrastructure de test
- Automatisation des tests de validation, par exemple avec un framework comme Appium

Lors du déploiement en production (App Store), il reste néanmoins nécessaire de suivre la qualité fournie aux utilisateurs avec des outils et techniques exposées dans la dernière partie de ce livre blanc.

## Les tests de charge

Le test de charge (ou stress) est une variante du test de performance, effectué en parallélisant un grand nombre de clients qui vont réaliser des transactions dans l'application.

Les objectifs sont multiples :

- Avoir une idée de la capacité maximum de la plateforme
- Identifier les goulots d'étranglement pour les réduire dans un deuxième temps

Dans le cas d'une application web, le test de charge peut être réalisé facilement depuis une infrastructure localisée en cœur de réseau pour simuler les requêtes HTTP, si nécessaire avec le 'user agent' mobile. ip-label possède par exemple sa propre infrastructure de génération de charge mise à disposition temporairement pour la durée du test.

Dans le cas d'une application mobile, l'infrastructure nécessaire pour émuler un grand nombre de terminaux 'clients' mobiles peut être couteuse. On peut alors solliciter les API utilisées à l'intérieur de l'application pour stresser le 'backend' avec de nombreuses requêtes HTTP, comme pour un site web.


# N°4 Mise en production |

# La mise en production

## La notion de “store” ou “market”

Pour pouvoir diffuser son application dans le monde entier ou sur une région spécifique, les concepteurs des systèmes d’exploitation mobiles comme Apple et Google ont créé la notion de « Store » d’applications.

Les utilisateurs Apple peuvent accéder à l’ « App Store » qui est passé de 500 Apps lors de son lancement en 2008 à plusieurs millions d’Apps aujourd’hui.

Google a suivi en lancant « Google Play » en 2012 qui a connu une croissance comparable. Le store de Google s’agrandit en permanence avec l’ajout de plus de 3000 nouvelles Apps chaque jour !

L’app store permet de gérer les paiements et les abonnements à une application, ainsi que la distribution des mises à jour. Il dispose également de fonctionnalités de suivi de l’utilisation par des statistiques sur les utilisateurs réels.

Cette facilité de gestion passe néanmoins par une rétribution de Apple ou Google en retour.

## Publier son application mobile

Publier son application permet de rendre visible et téléchargeable son app pour tous les utilisateurs de ce store. Il existe un certain nombre de vérifications et de restrictions posées par Google et Apple pour que votre application soit acceptée et postée sur ces stores.

### Android

Pour Google par exemple, aucun process de validation n’est mis en place pour toute application postée, ce qui explique le temps assez court de la visibilité des applications sur « Google Play » qui est de 2 heures approximativement. Google fait seulement une vérification des licences pour s’assurer du respect des droits d’auteurs.

Google a néanmoins mis en place un système appelé « Bouncer » qui s'occupe de scanner toutes les applications postées sur son store pour détecter et supprimer toutes les applications qui posent un problème de sécurité. Pour les normes de codage, Google n'impose aucune règle spécifique, et donc libre aux développeurs de faire comme ils le souhaitent !

Pour pouvoir publier son application sur le magasin d'application de Google « Google Play », chaque développeur a besoin d'abord de souscrire au programme « Google Developer », un abonnement annuel. Après avoir eu l'accès à ce programme, chaque développeur a la possibilité de poster un nombre infini d'applications sur le store de Google.

## iOS

Apple, contrairement à Google, a un processus de validation beaucoup plus strict, Apple a un contrôle total sur tout le contenu de son App Store. Une nouvelle application postée sur le store d'Apple peut prendre de 1 à 3 semaines pour être validée tandis que ce processus prendra de 2 à 10 jours ouvrés en cas de mise à jour d'une application existante.

Apple impose aux développeurs d'application iOS d'utiliser seulement les bibliothèques de codes et le SDK officiel d'Apple. Toute utilisation d'une API privée entraine le rejet de l'application lors du processus de validation. D'autres facteurs de rejet sont aussi à noter comme les applications ayant un contenu ne respectant pas les droits d'auteurs, contenant de la discrimination ou de la pornographie...

### ⊕ Les plus d'une publication sur l'App store

- Une distribution et visibilité à des millions d'utilisateurs dans tout le monde, avec une possibilité de ciblage par zone géographique
- Un modèle commercial permettant des options payantes ou un abonnement récurrent via le store (modulo les royalties Apple ).

### ⊖ Les moins

- La recherche sur le store est néanmoins moins performante pour l'acquisition de trafic que celle via un moteur de recherche web.

# Suivi statistique et analytique

Une fois que le site web ou l'application mobile est en production, le suivi des utilisateurs et l'analyse des données qui en découlent permettent de répondre à 2 catégories de besoins :

- **Marketing** : analyse du comportement des utilisateurs (tunnel d'achat..), segmentation, etc. Les données telles que le nombre de téléchargements, le nombre d'utilisateurs ou les pages les plus consultées sont des données essentielles pour comprendre l'usage réel de votre application.
- **Technique et troubleshooting** : identifier les causes de lenteurs et tout problème technique que les utilisateurs rencontrent pendant l'utilisation

## Analytique des sites web mobiles

Pour le web, les solutions de suivi des utilisateurs sont techniquement implémentées par javascripts et cookies HTTP permettant :

- De reconstituter un parcours utilisateur
- De conserver le suivi dans le temps d'un utilisateur : nouveau visiteur ou récurrent ?
- De suivre des indicateurs métiers comme le "taux de rebond" (Bounce rate)

## Analytique des apps mobiles

Il existe trois grandes catégories d'analytique pour les applications mobiles :

- **App Marketing Analytics** : Sert à déduire la visibilité de votre application sur les stores, le nombre de téléchargements et d'utilisation.
- **In-App Analytics** : Sert à analyser comment les utilisateurs réagissent avec votre application, quelles pages consultent-ils, etc…
- **Crash & Performance Analytics** : Sert à analyser la disponibilité de votre application, à détecter les évènements de "crash" en les catégorisant par type de devices etc… Pour analyser les lenteurs et le fonctionnement interne de l'application, on parlera dans ce livre blanc d'outils de 'troubleshooting'.

Parmi les outils les plus connus servant à faire de l'analytique pour les applications mobiles, on peut citer Google Analytics (Web et applications mobiles), Mixpanel, Flurry, Fabric (ancien Crashlytics…).

Certains outils couvrent à la fois le web et les applications mobiles (console de pilotage unique) et nous allons donc les rassembler dans un seul chapitre ci-dessous.

## Les outils Analytics

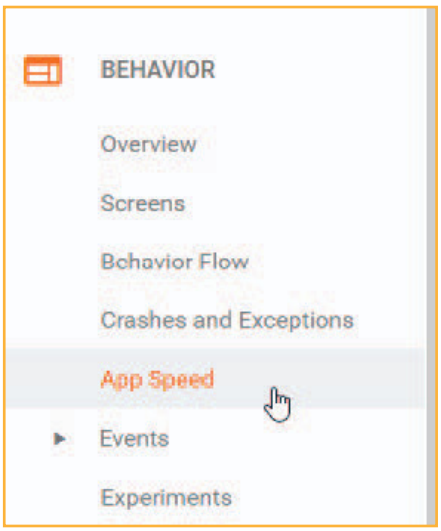
Ils se déclinent par un code embarqué (Javascript, souvent via un outil de tag management) en web ou app mobiles (SDK).

### Google Analytics (web & app mobile)

Google Analytics (https://www.google.com/analytics/) est un outil gratuit pour les fonctionnalités de base, qui apporte des informations sur le comportement des utilisateurs dans un objectif Marketing (conversion du trafic en actions monétisables) :

- **Acquisition /Audience** : Il est possible de distinguer les nouveaux utilisateurs de ceux déjà inscrits, avec des informations sur leur pays, leur langue, la version de leur application.
- **Comportement/Engagement** : créer des actions pour tracker vos utilisateurs et générer des rapports.
- **Conversion** : Fixer des objectifs, tracker votre conversion en fonction de ces objectifs puis analyser les résultats.

Pour les applications mobiles, des fonctionnalités de 'troubleshooting' sont par ailleurs disponibles dans les sous menus 'Crash et exceptions' et 'App speed', ce dernier donnant accès à des métriques qui doivent être codées explicitement et qui ne sont donc pas disponibles par défaut :




La solution Google « Firebase Performance Analytics » fonctionne sur le même principe d'instrumentation dédiée dans l'application.

## Apple App Analytics (iTunes Connect)

Disponible depuis 2015, la solution analytics d'Apple est gratuite et offre 3 outils pour analyser vos données :

1. Données de l'App Store
2. Données sur les ventes
3. Données sur les usages

Cette fonctionnalité offerte par Apple permet de contrôler le nombre d'achats, installations et utilisations de votre application par zone géographique. Vous pouvez aussi avoir le nombre de vues de votre application dans l'Apple Store et déterminer si les utilisateurs qui ont installé votre application l'ont trouvé en navigant dans l'Apple Store, à partir d'une recherche sur ce dernier, à partir d'une application ou site web référent....

Lors de la sortie d'une nouvelle version d'iOS, le développeur peut identifier par exemple une diminution du nombre de téléchargements de son application si elle n'est pas compatible avec cette nouvelle version.

Apple Analytics vous propose aussi un service de rapports hebdomadaires envoyés par email contenant les principaux indicateurs fournis : nombre d'impressions, nombre d'installations, achats, nombre de sessions et nombre de pannes/crashs (pour chaque application).

## Les outils de troubleshooting

En plus du suivi du comportement des utilisateurs (analytics), **il est nécessaire de pouvoir identifier et diagnostiquer les dysfonctionnements :**

- Crash
- Erreurs applicatives et sur les API
- Performances et lenteurs constatées (voir chapitre 4)

On a vu que les outils de suivi d'audience permettent d'avoir une première visibilité sur les Crash, mais il est nécessaire d'utiliser une solution dédiée si on désire suivre les erreurs applicatives détaillées ou pour les appels externes de type 'API' (services tiers de recherche, notifications...) plus complexes à capturer et restituer.

### Troubleshooting d'une application mobile

Un outil de troubleshooting s'insère pour une application mobile comme une librairie tierce (SDK) au moment de la compilation. A l'exécution sur le terminal de chaque utilisateur, celle-ci capture automatiquement les dysfonctionnements applicatifs et des API externes appelées par le code.

**Crashlytics** permet de surveiller les rapports de crash (les notifications des pannes) que subit votre application. Ces informations sont disponibles depuis l'interface du site web, mais aussi depuis une application dédiée. Depuis le rachat par Fabric, l'outil est intégré dans un framework plus large de développement.

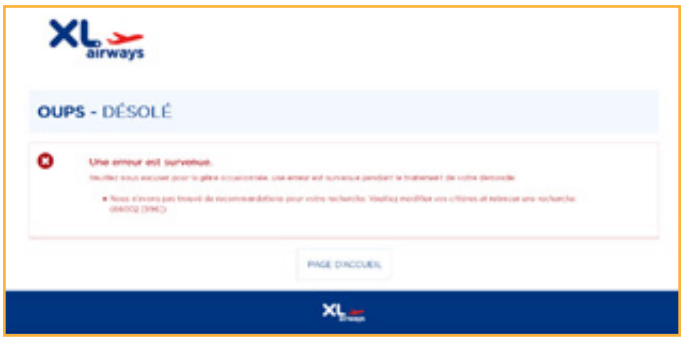
Un autre exemple de SDK orienté performance est **Firebase Performance Analytics**, qui permet de capturer des temps de réponse par chronos start/stop.

## Troubleshooting d'un site web mobile

Les applications Web ou les applications multi-plateformes utilisent des pages HTML et javascript pour l'affichage à l'utilisateur. Dans le cas des applications multi-plateformes, c'est un navigateur embarqué qui affiche la page (WEBVIEW). Le suivi de la performance des web views est également possible via un code embarqué Javascript utilisant l'API des Navigations Timings du W3C :

- Temps DNS et TCP (réseau)
- Temps de réponse du serveur
- Temps de chargement

Il est également possible de suivre des actions sur mesure avec la technologie RUM de Ekara Web via des chronomètres start/stop.




Pour suivre les erreurs sur une page web mobile, il est par exemple possible de capturer les messages standards générés par l'application, notamment les codes erreurs métier. En effet, ceux-ci sont générés par l'application et intégrés dans le code HTML de la page, ce qui les rend exploitables par un outil dédié à cette fin.

Ceci permet ensuite de suivre l'évolution ou la répartition des erreurs par type (ou code métier).

L'interprétation de ces erreurs dépend de la typologie définie dans l'application métier.


# N°5 Performance |

# Et la performance dans tout ça ?

**La performance est une composante importante du parcours “digital” des utilisateurs.** En effet, les études récentes comme celle de Doubleclick montrent que le chargement complet d’un écran corrèle fortement avec les indicateurs marketing de conversion. Plus de la moitié (53%) des utilisateurs quittent un site web mobile qui ne s’est pas chargé en 3 secondes.

L’objectif est **d’optimiser cette performance afin d’augmenter l’atteinte des objectifs marketing** comme le taux de rebond (bounce rate) ou les conversions. Nous verrons dans les chapitres suivants les indicateurs permettant de formaliser ces objectifs et de suivre les actions d’améliorations dans le temps.

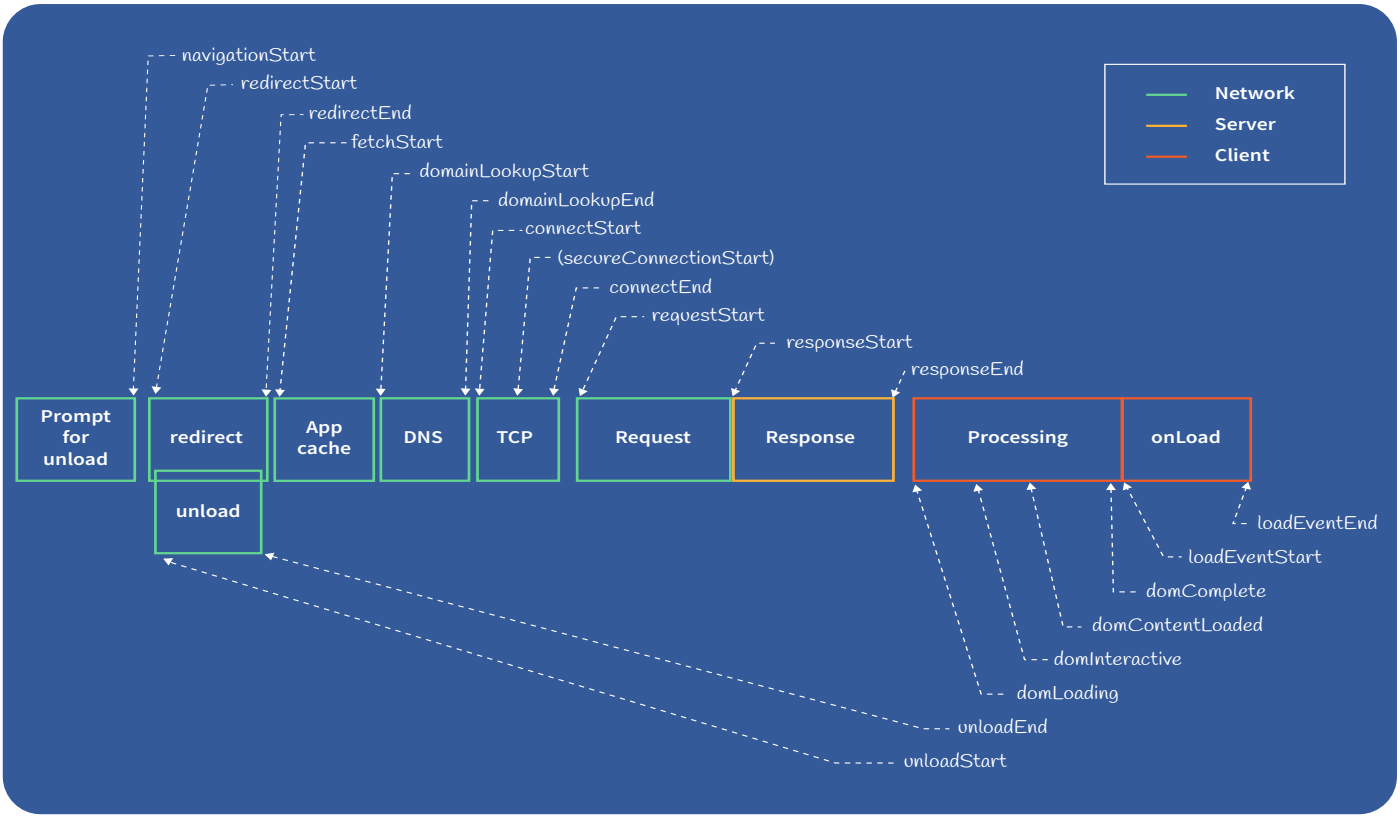
## Quels sont les critères de performance ?

Le suivi d’indicateurs de performance est nécessaire pour identifier les utilisateurs rencontrant des problèmes techniques et des lenteurs.

Les métriques de performances s’adaptent aux types d’application : sites web ou app natives (iOS/Android).

### La performance des sites web mobiles

Pour les **sites web (responsives ou dédiés aux mobiles)** : nous conseillons de mesurer la fin de chargement et la possibilité d’interagir avec la page avec la métrique ‘**Time To Interact**’ normalisée par le W3C et disponible via l’API Navigation Timing des navigateurs récents. Elle permet de savoir quand l’utilisateur est capable d’interagir avec la page (clic, scroll..). La métrique TTI est nommée “domInteractive” dans le schéma ci-dessous du W3C :



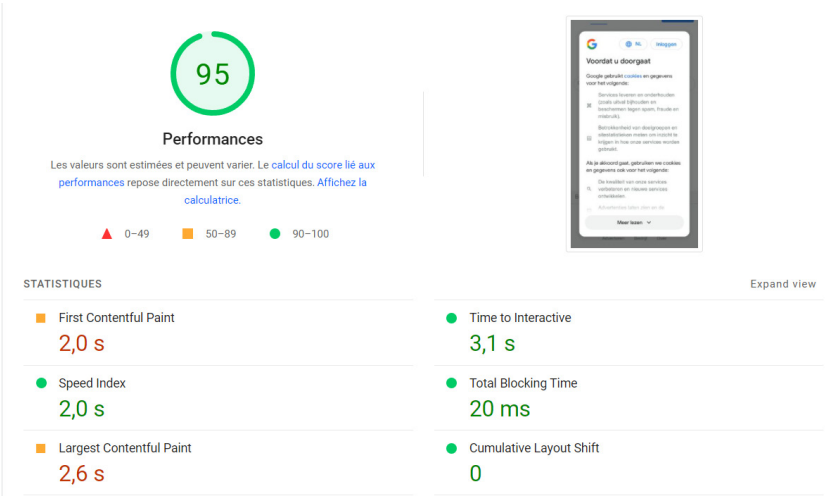

*Source : https://www.w3.org/TR/navigation-timing/*

Par ailleurs, le **Speed Index** est devenu un standard car **il permet de déterminer si l'affichage visuel est rapidement complet en mesurant la progression de l'affichage d'une page.** Il calcule un score global au plus près de l'expérience vécue par l'utilisateur. La valeur obtenue est un index qui doit se comparer avec d'autres pages web. Il est donc particulièrement utile :

- lorsque l'on compare son site à celui de la concurrence
- pour comparer l'effet des optimisations sur une page lors d'un test A/B.

## Outils disponibles gratuits

Google propose l'outil **Page Speed Insights qui note les performances fixes et mobiles d'un site web.** L'outil prend en compte les **Google Core Web Vitals.**

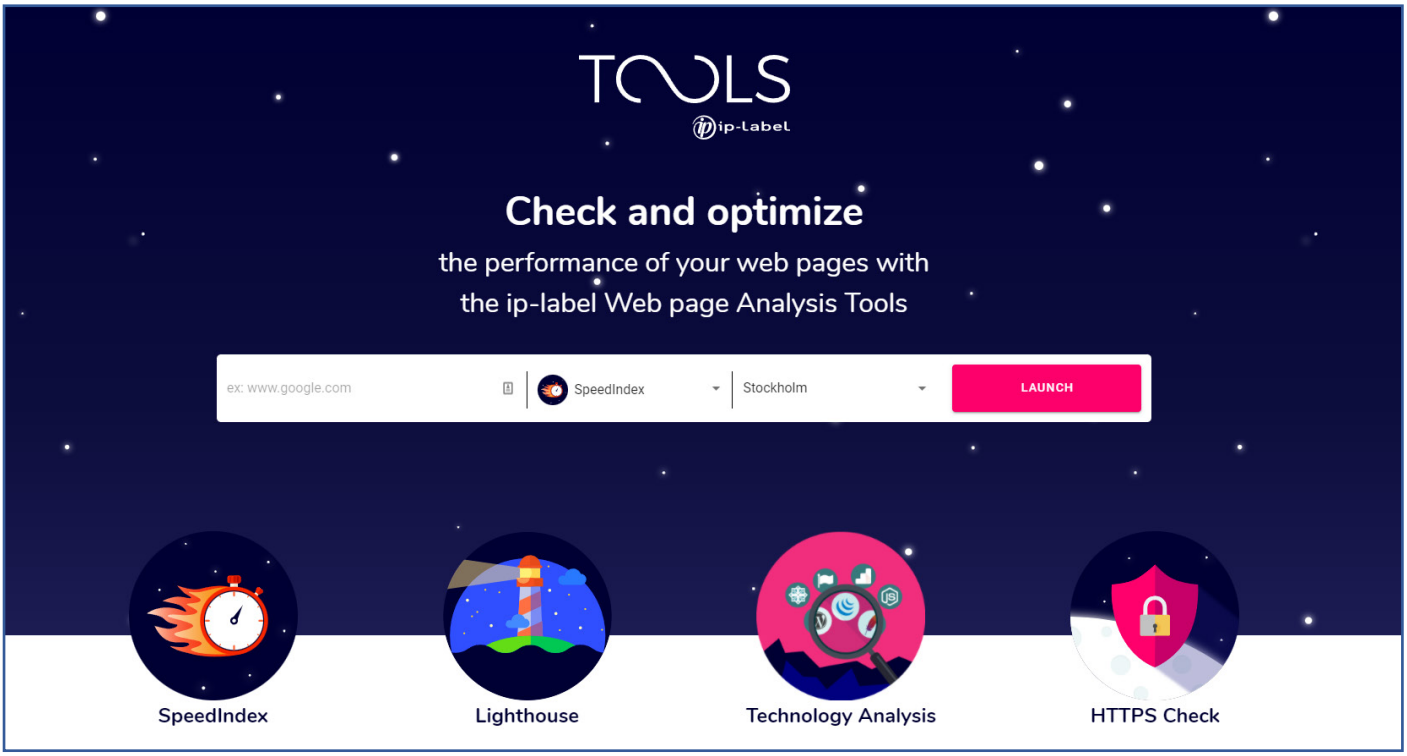



*Source : https://pagespeed.web.dev/*

On note que depuis récemment, les indicateur “FCP” (First content paint) et “DCL” (DOM content loaded) issus de l’API W3C de Navigation timing sur les navigateurs Chrome d’utilisateurs réels sont intégrés dans la note et le résultat.

ip-label propose également des outils d’analyse automatique (Google Lighthouse, etc.) depuis un environnement de cœur de réseau calibré et dédié aux tests: http://tools.ip-label.io. A noter que Lighthouse est également intégré dans Chrome Dev Tools pour les tests manuels.

Le site https://tools.ip-label.io permet de calculer notamment avec Lighthouse le “**Speed Index**” :

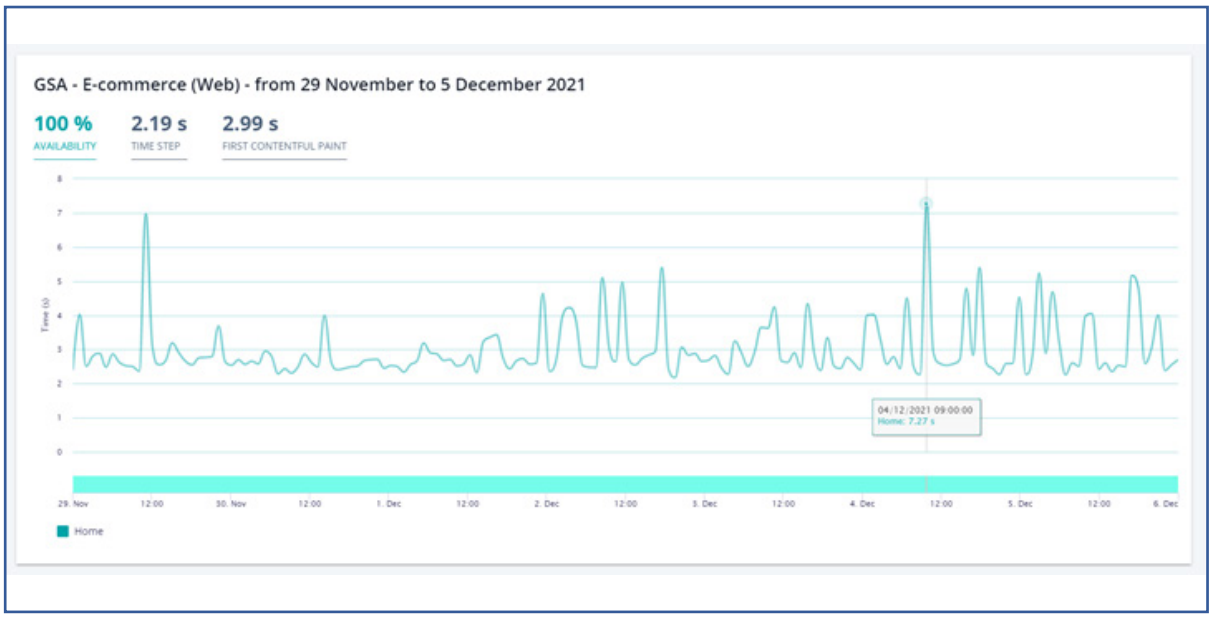



## Mesures robots ou automatisées

Un test manuel étant par définition ponctuel et une page web étant amenée à évoluer rapidement, il est indispensable d’automatiser la prise de mesure de performance pour la suivre dans le temps.

Un robot peut mesurer dans un contexte stable la métrique de chargement de page web ou d’écran d’une application mobile (validation visuelle).

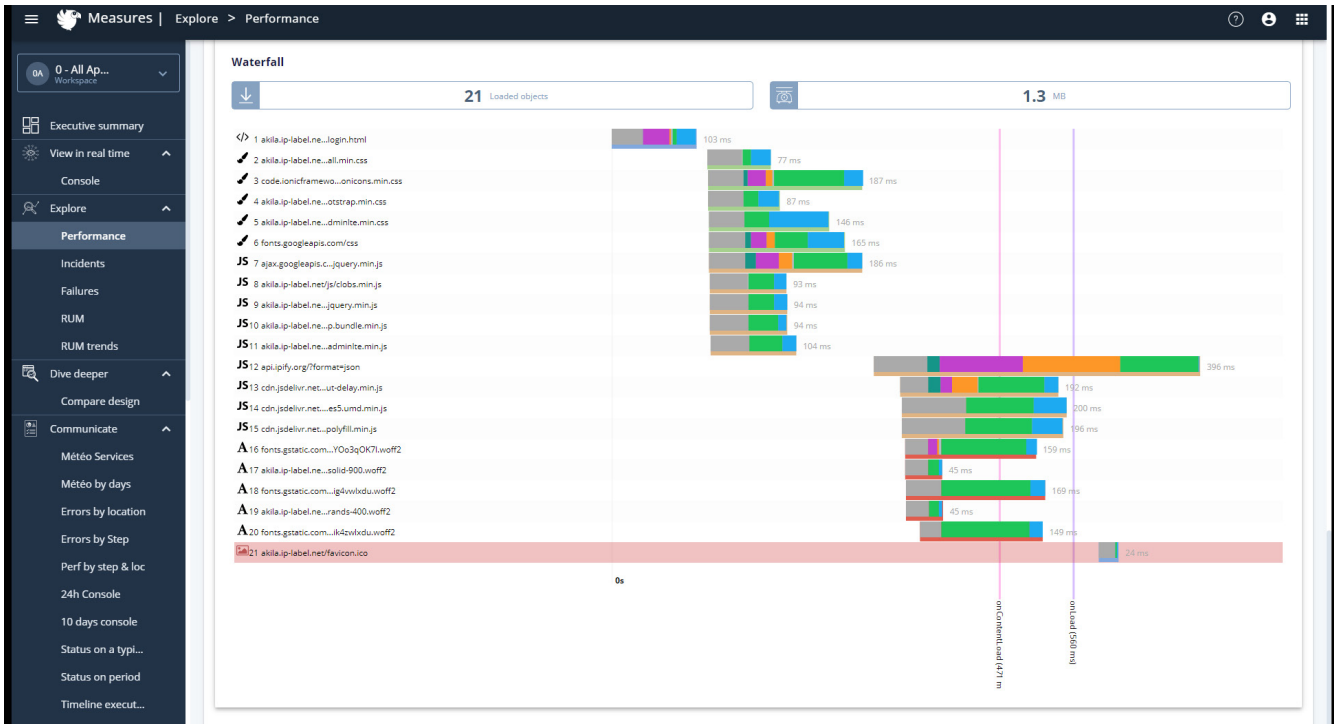
L’exemple ci-dessous montre les différents temps de **page Web** (Time to Interact, chargement de page...) mesurés depuis un Robot en cœur de réseau Internet (si nécessaire avec une émulation du user agent, de la bande passante mobile et la taille d’ecran) :




*Source : ip-label*

**Le LCP (Largest Contentful Paint) permet** permet en général de **mieux évaluer l'impact visuel d'une lenteur pour l'utilisateur.**

Le monitoring robot permet également une analyse de contenu détaillée comme on le voit dans le rapport ci-dessous :




*Source : ip-label*

En plus du temps de réponse technique, les bonnes pratiques de codage et des indicateurs de performance peuvent être évalués sous forme de 'Score' par des outils comme Google Lighthouse ou très simplement avec l'outil "Mobile friendly" (https://search.google.com/test/mobile-friendly)

Concernant le **Speed Index**, l'avantage de l'utilisation d'un robot pour le mesurer est de pouvoir apprécier sa variation dans le temps suivant les mises à jour. Un robot peut également émuler un terminal mobile pour mesurer la version responsive d'un site web.

## Mesures Web Real User Monitoring (code embarqué)

En plus des métriques natives de chargement des pages web complètes, avec **Ekara RUM**, il est possible de définir des chronomètres personnalisés. Ceci vous permet de mesurer le **chargement d'un composant spécifique** et plus largement de mesurer la durée entre deux évènements choisis par vos soins.

Les applications Web ou les applications multi-plateformes utilisent des pages HTML et javascript pour l'affichage dans un navigateur qui supporte la mesure de performance **Navigations Timings** du W3C.

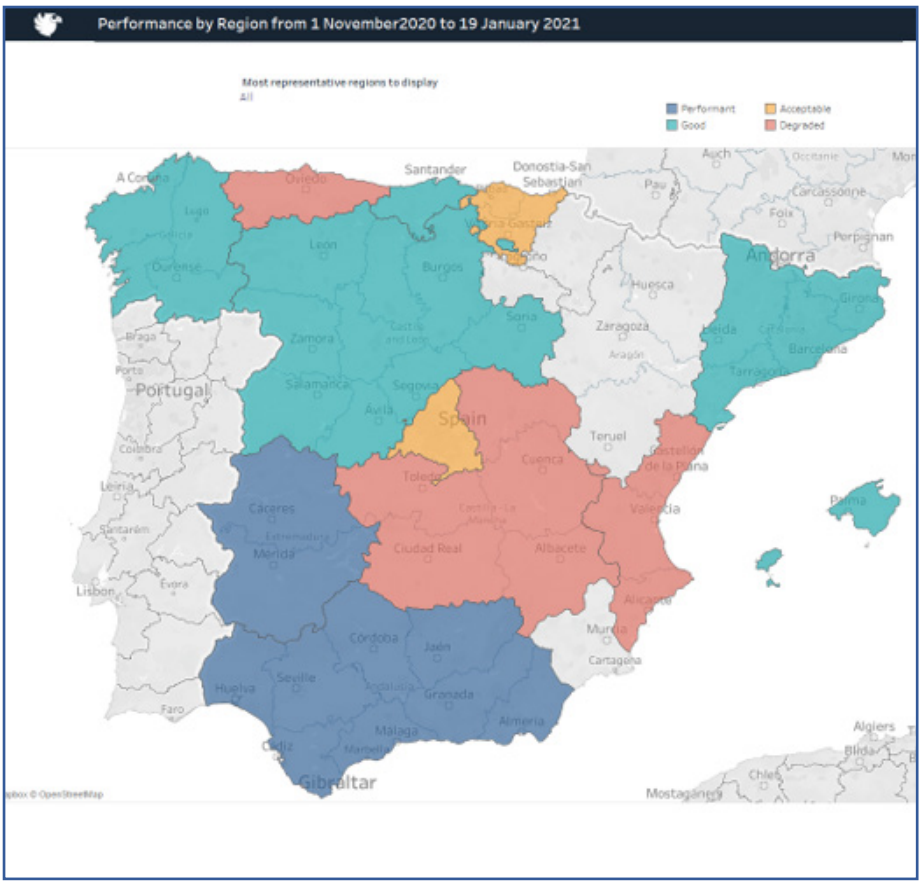
Le suivi de la performance des navigateurs ou des web views sur des utilisateurs réels est donc possible via un code embarqué Javascript utilisant l'API des Navigations Timings du W3C qui permet de mesurer :

- Temps DNS et TCP (réseau)
- Temps de réponse du serveur
- Time to interact (TTI)
- Temps de chargement

**Il est important néanmoins de corréler la performance avec des informations métiers plus complètes.** Par exemple, ip-label a la notion de "custom dimension" pour capturer en même temps des informations pertinentes :

- Localisation (y compris dans un réseau privé)
- Utilisateur concerné (identifiant)
- Version de l'application
- Référence du serveur utilisé lors de la mesure
- Navigateur et Terminal utilisé : fixe ou mobile, marque, modèle, version d'OS
- …

**Une solution de Datavisualisation est indispensable pour analyser ces dimensions au cas par cas.** Par exemple, la notion de géolocalisation (via adresse IP) permet d'identifier les zones moins bien desservies, comme dans le graphique ci-dessous pour l'Espagne :




*Source : ip-label Ekara Mobile*

## La performance des applications mobiles

Pour les **applications natives**, il n'existe pas de norme technique pour mesurer le chargement visuel de l'écran. Nous conseillons de différencier :

- le **temps passé sur l'écran** par l'utilisateur (Temps "screen Display"). C'est une métrique qui se rapproche de près du comportement utilisateur, mais qui inclut le temps d'inactivité ou de lecture des informations.
- le **temps de chargement initial de l'écran**. Comme celui-ci est constitué souvent de plusieurs modules (Classe principale, webviews...), il est difficile à reconstituer. On peut alors recourir à une métrique 'sur mesure' positionnée explicitement par le développeur comme un chronomètre aux endroits du code qui sont pertinents.
- la **fluidité de rafraichissement pendant l'utilisation** peut être mesurée en images par seconde (ou FPS = frame per second). L'idéal de performance est de 60 FPS pour une impression de fluidité maximale (par exemple dans le scroll des listes déroulantes).

## Outils gratuits

Des outils de mesure de performance existent sous Android Studio comme **Traceview** : https://developer.android.com/studio/profile/traceview.html

L'outil permet d'identifier les méthodes du code qui occupent le plus le CPU comme montré ci-dessous :

| 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/lan... | 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% |

Toujours sous Android, on peut mentionner l'outil de "Profiling" **Hierarchy Viewer** qui permet de mesurer le temps d'affichage des vues (Draw time) : https://developer.android.com/studio/profile/hierarchy-viewer.html

Sous iOS, l'outil de profiling équivalent intégré dans X-Code est "**Instruments**".

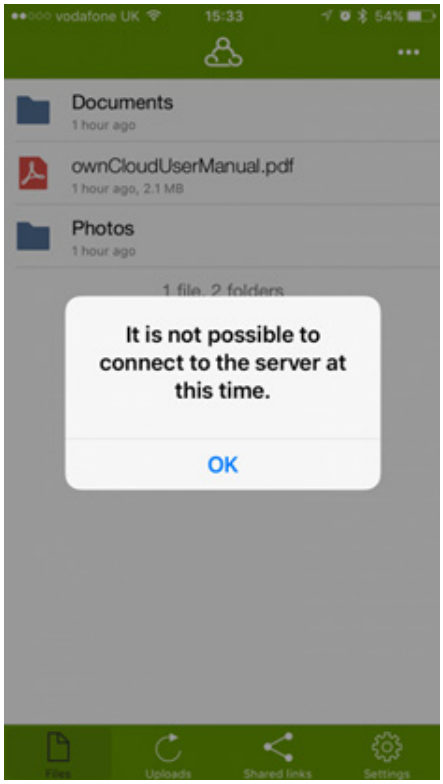
Ces outils demandent néanmoins une expertise avancée pour obtenir des métriques pertinentes.

## Les mesures robots des applications mobiles

**La mesure automatisée** effectuée par un robot **permet de restituer le temps passé sur l'écran par un utilisateur de test**, avec des contrôles visuels qui peuvent etre décomposés en plusieurs étapes d'une transaction.

L'avantage de la mesure robot est qu'elle n'est pas dépendante du framework utilisé pour développer l'application mobile.

En cas de perturbation, on pourra identifier, par capture d'écran, l'étape de la transaction qui a bloqué comme le montre l'image ci-contre :




La caractéristique de la mesure d'un robot est également de pouvoir contrôler un affichage visuellement, donc de prendre en compte tous les chargements techniques inclus dans un écran (webviews, etc.) pour vérifier l'**intégrité d'une transaction**.

Par ailleurs, les applications natives incluent souvent des '**webviews**' qui sont des "mini navigateurs web" embarqués et qui permettent donc de remonter des métriques orientées "Web" et issues des utilisateurs réels (RUM) comme le TTI (Time To Interact).

## Le robot mobile virtuel Android : Cloud Android

L'offre Ekara Mobile vous permet de disposer de terminaux Mobile Android dans le cloud. Vous pouvez ainsi virtualiser les applications installées.

**Cette solution permet de déployer plus facilement des variantes de configuration Android suivant les besoins.** Vous pouvez réaliser des mesures de performance pour des marchés régionaux (Europe, USA, etc.) ou pour des versions systèmes différentes (Android 9/10/11).

# Applications Web ou natives : actif + passif, une stratégie gagnante

Aujourd'hui, une solution de gestion de la performance applicative doit inclure 2 aspects importants :

1. Le suivi de la **disponibilité du service 24h/24**, par l'intermédiaire de robots de test qui vont simuler une transaction utilisateur. Lors d'un ralentissement, le contexte stable d'un robot permettra d'objectiver le ressenti réel des utilisateurs (affichage visuel ou Speed Index).

   Il est possible d'utiliser des **terminaux réels** pour l'exécution des tests sur iOS ou Android. Sur Android, un **terminal virtuel Cloud** peut être envisagé pour réduire les coûts de déploiement.

2. Le **suivi des utilisateurs réels (RUM)** fait via un code embarqué (Javascript ou SDK) dédié à la collecte d'indicateurs de performance.

# L'évolution des bonnes pratiques de sécurité mobile

Comme pour l'ensemble de l'économie digitale, le mobile s'adapte aux nouvelles exigences de sécurité. L'authentification par login et mot de passe est devenue insuffisante.

La bonne pratique actuelle consiste à utiliser d'autres preuves d'identité en complément. C'est le cas quand une transaction mobile vous impose un code SMS ou vous invite à valider votre action par un clic d'authentification dans l'application mobile présente sur votre téléphone. On parle alors de **MFA ou authentification multi-facteurs (**"MFA" en anglais).

Les meilleures pratiques vont encore plus loin. Les technologies MFA imposent des étapes supplémentaires dans le parcours client et peuvent conduire en cas de problème à des abandons et donc de la perte de chiffre d'affaires.

D'autres méthodes d'authentification sont disponibles ayant pour avantage d'être moins intrusives et donc de réduire au maximum la "friction". Les méthodes les plus en vue actuellement sont regroupées sous le terme de sécurité basée sur les comportements ("Behaviour Based security"). Ce concept regroupe des techniques diverses pour vérifier que l'utilisateur est légitime. On y retrouve les outils de type "captcha" ou les détections d'actions "forcément humaines" comme le  déverrouillage d'un téléphone. En résumé, toutes les solutions de détection de robots.

La réglementation internationale évolue rapidement sous la pression du risque de sécurité. L'un des exemples récents concerne la règlementation bancaire européenne qui prévoit dans sa norme DSP2 la généralisation du MFA.

La mise en place d'outil de détection de Robots impose aussi aux offres de service de monitoring de trouver des parades technologiques pour continuer à assurer le service : savoir déverrouiller un téléphone, passer un captcha avec succès...
Enfin, le respect des conditions d'utilisation des éditeurs de logiciel réduit d'autant les possibilités techniques de pilotage des terminaux. Impossible de proposer une solution basée sur des techniques de routage sur Android. Illusoire de compter sur un Jailbreak iOS aujourd'hui.

# Conclusion

Dans la plupart des pays, la tendance est forte vers un web mobile. Dans certains domaines comme le e-commerce, une majorité de pays connait déjà un pourcentage de transactions plus fort sur les applications mobiles que sur le web desktop.

Au-delà des aspects techniques, le choix de conception entre des sites web 'responsives' uniques pour desktop/mobiles/tablettes et un site (ou une application) dédié par environnement doit aussi prendre en compte la complexité des produits. En effet, un produit complexe nécessitera probablement un 'tunnel' ou 'processus' d'achat dédié par environnement pour optimiser le processus d'achat sur celui-ci. La notion de « Mobile First » qui consiste à offrir à l'utilisateur une véritable expérience mobile dédiée prend alors tout son sens.

Quel que soit le choix technologique, pour chaque environnement, on a vu que des solutions de monitoring existent pour corréler les indicateurs business avec le fonctionnement technique.

Nous espérons que ce livre blanc contribuera à aider les équipes Marketing et Développement à se mettre d'accord sur des indicateurs afin d'améliorer significativement la performance applicative et les résultats métiers associés.

# A propos du groupe ip-label

**Depuis 2001, ip-label - éditeur de logiciels dans l'eXpérience Utilisateur - aide les entreprises, dans plus de 25 pays, à gérer et à optimiser la performance de leurs applications critiques (sites web, applications métiers, mobiles, voix, téléphonie, etc.). Son offre centrée sur l'expérience utilisateur permet d'améliorer la disponibilité et les temps de réponse des applications, moteurs du succès des entreprises, leur permettant d'accroître in fine leur audience, leurs revenus et leur productivité.**

Vous souhaitez en savoir plus sur la gestion de l'expérience digitale ?

**Demandez une démo**

ekara by ip-label

ekara.ip-label.com
+33(0)1 77 49 53 00
marketing@ip-label.com

Paris I Madrid I Stockholm I Helsinki I Shanghai I Tunis