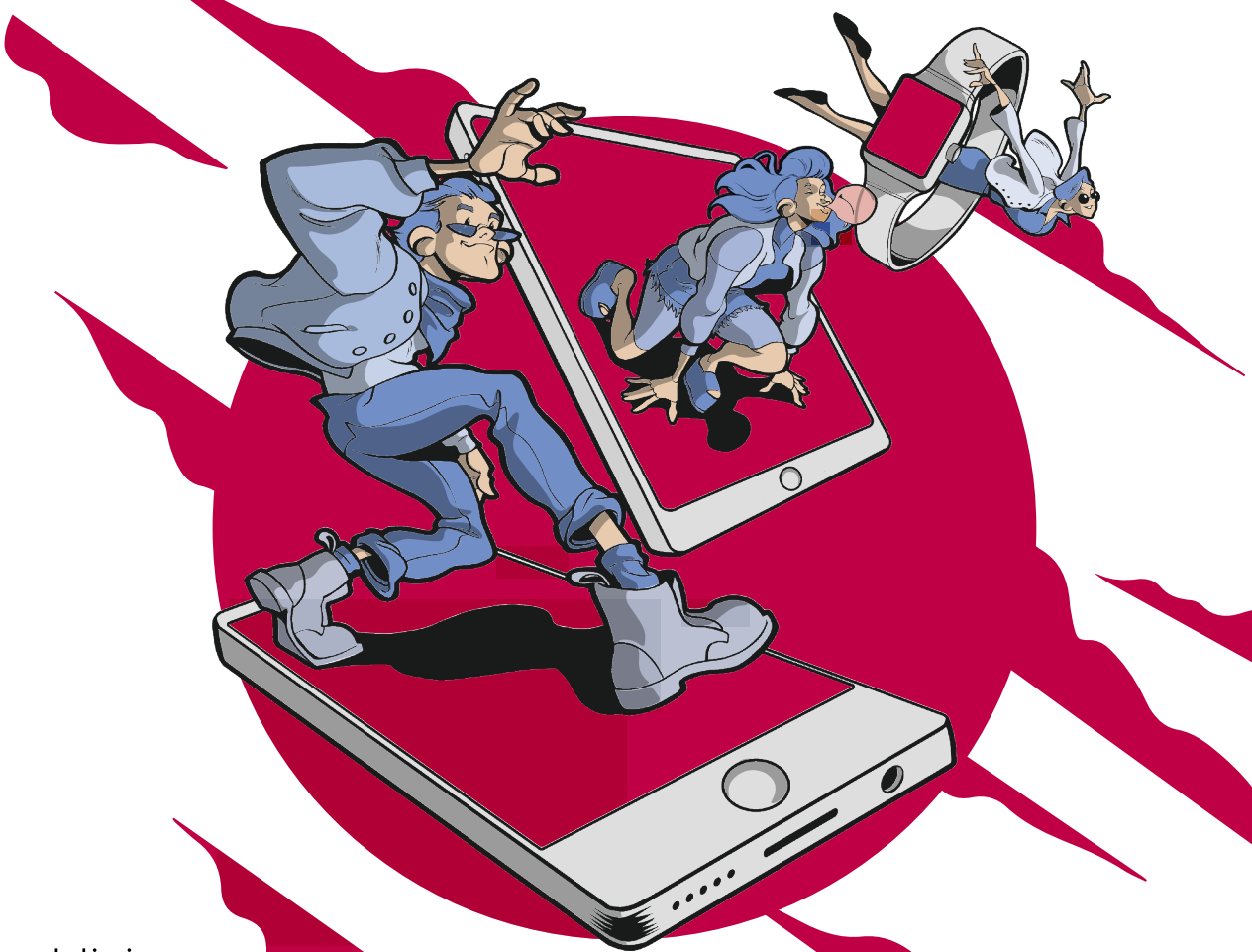
TechTrends #8

Le Mobile aujourd'hui : de la réalisation à l'évolution

Comprendre, Façonner, Innover

BY ENGINEERS *at* PUBLICIS SAPIENT



publicis
sapient

Les TechTrends sont l'expression de notre savoir-faire ; forgé sur le terrain, auprès de nos clients dans le cadre des projets que nous menons avec eux.

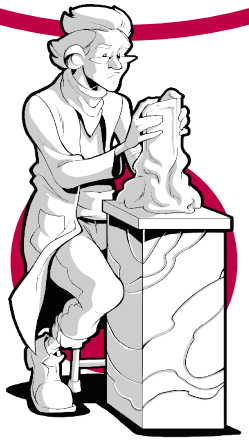
Fruit d'un travail collaboratif de nos consultants, vous y trouverez, nous l'espérons, les nouvelles tendances technologiques et méthodologiques ainsi que l'état de l'art de notre profession.

Nous tentons, dans le cadre de ces publications, de vous dispenser des conseils directement opérationnels afin de vous guider dans les décisions stratégiques que vous avez à prendre.

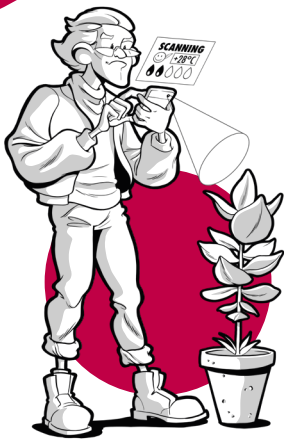
Distribués à plusieurs milliers d'exemplaires tous les ans, la collection des TechTrends s'étoffe régulièrement de nouveaux ouvrages.

Bonne lecture !

Comprendre



Façonner



Innover

Introduction

Dès 1920, les scientifiques[1] ont commencé à décrire un monde où des terminaux portables seraient au centre de nos activités quotidiennes.

Aujourd'hui, nous pouvons dire que ces visionnaires étaient dans le vrai : les dispositifs nomades accompagnent indéniablement notre quotidien. Pour gérer nos vies professionnelles ou personnelles, pour répondre à un message, pour éteindre l'éclairage de nos habitations, pour reconnaître nos amis sur des photos ou pour traduire en temps réel des panneaux en kanji, notre smartphone est là, prêt à répondre à nos besoins - pourvu que connectivité et batterie ne nous aient pas abandonnés.

La transition a été graduelle et pourtant rapide : 7 ans à peine séparent le Motorola MicroTAC de 1992, qui ne fournissait même pas un carnet d'adresse numérique, du Palm VII, capable de naviguer sur le Web. L'iPhone - et iPhone OS - débarquent en 2007 rapidement suivis par le T-Mobile G1, le premier téléphone grand public à exécuter une version du système d'exploitation Android.

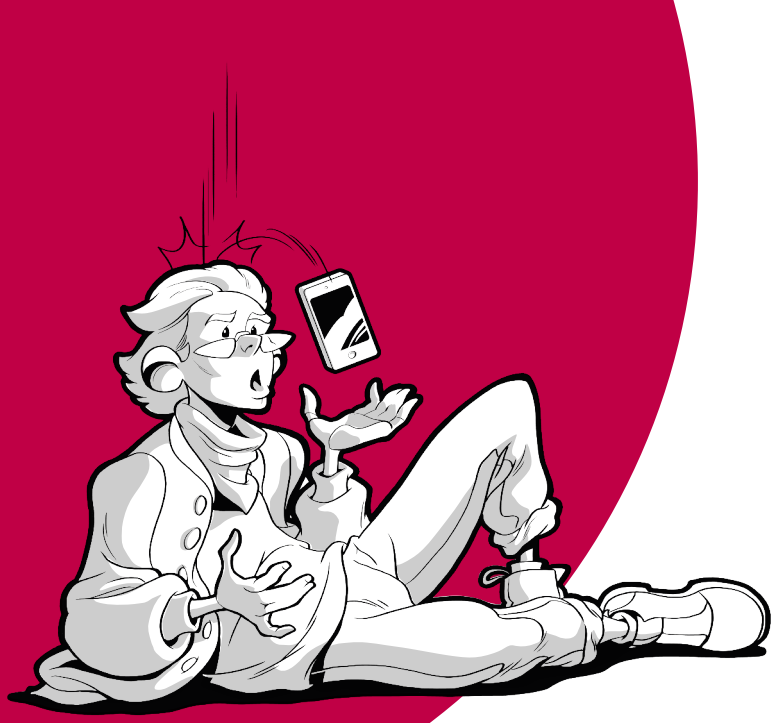
Si, d'un côté, il est incontestable que la miniaturisation des composants a été l'élément déclencheur de cette véritable révolution, c'est le logiciel qui a rendu possible la transition et surtout **l'adoption massive par les utilisateurs** grâce à des interfaces intuitives, fluides et capables de réussir là où les agendas numériques des années 90 avaient échoué.

L'évolution du logiciel a logiquement provoqué **l'évolution du métier de développeur mobile** qui est passé de simple réalisateur d'interfaces statiques à véritable ingénieur capable de répondre aux complexités du domaine : **conception d'APIs optimisées, création d'applications modulaires, design de composants réutilisables** ou encore **mise en place de stratégies de tests robustes**.

Le smartphone étant désormais le centre de nos vies connectées et les applications constituant au moins 80% de son utilisation, **réaliser un logiciel de qualité** est aujourd'hui plus important que jamais. Il devient donc primordial de **comprendre les complexités que cela implique** et de s'appuyer sur une équipe capable de les adresser efficacement.

Dans ce TechTrends, évolution d'une première édition publiée en 2015, nous vous proposons un **éclairage sur les technologies et les pratiques qui accompagnent un développement mobile de qualité** et notre vision sur le futur des domaines qui lui sont associés.

1. http://www.businessinsider.fr/us/tesla-predicted-smartphones-in-1926-2015-7 (https://bit.ly/2kxvn9n)



Comprendre

Le monde est mobile, la question ne se pose plus. Les cas d'usage sont de plus en plus nombreux.

Notre smartphone est à la fois le point d'accès à nos finances, à nos magasins préférés, à l'information et à nos données de santé. Nous nous en servons, par exemple, pour commander un taxi, nous identifier lors d'un trajet en train ou en avion ou encore suivre nos progrès dans nos activités sportives…

Plus récemment, le terminal nomade est devenu le **centre de contrôle de notre vie numérique.** Nos smartphones sont désormais conçus pour s'interfacer avec montres connectées, dispositifs médicaux, véhicules personnels et systèmes domotiques. Paradoxalement, même l'usage d'un ordinateur devient de plus en plus subordonné au mobile.

Le smartphone a aujourd'hui perdu son statut d'appareil de communication pour finalement devenir le point d'accès à l'intégralité de notre vie numérique : **un véritable hub portable par définition**.

Le paradigme **Mobile First, qui était sur toutes les lèvres il y a quelques années, s'est d'ailleurs imposé comme un standard de facto.**

L'expérience utilisateur au cœur de l'adoption

" Les spécialistes de la User eXperience ont rapidement démontré que les parcours utilisateur sur les terminaux mobiles étaient très différents de ceux sur le Web et devaient donc être pensés spécifiquement. "

C'est le monde du design et de l'ergonomie qui a fait émerger la notion de Mobile First. Les spécialistes de la User eXperience ont rapidement démontré que **les parcours utilisateur sur les terminaux mobiles étaient très différents de ceux sur le Web** et devaient donc être pensés spécifiquement.

Déterminer les fonctionnalités clés de votre application

Un temps, la tendance était d'offrir toutes les fonctionnalités sur le Web et seulement une partie sur mobile. Ce déséquilibre tend aujourd'hui à s'inverser.

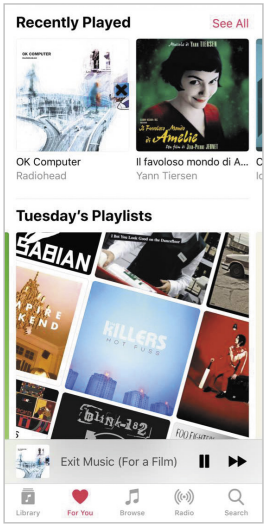
Néanmoins, sur mobile, l'attention de l'utilisateur est plus partielle. Il peut être distrait à tout instant, pour traverser la rue ou bien répondre à un appel. Il faut donc être plus direct : **l'utilisateur doit être capable de compléter un processus en un minimum d'étapes**. La navigation privilégiée sera linéaire et prendra en compte les zones de l'écran dites « chaudes » (c'est-à-dire facilement accessibles avec les doigts, en bas de l'écran sur mobile).

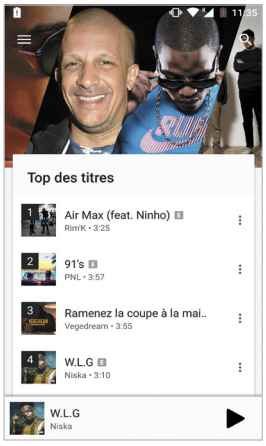
De ces différences d'usage est née **la notion de fonctionnalités clés**. Les ergonomes et les Product Owners doivent identifier quelles parties de leurs produits sont les plus critiques et doivent être mises en exergue sur le mobile. La fluidité du parcours mobile est aujourd'hui un point critique pour les producteurs d'applications car **une application mal pensée mènera à un abandon immédiat** par les utilisateurs, très volatiles.

Il devient donc nécessaire de **comprendre les usages et de récolter un maximum de retours utilisateurs**. Pour ce faire, les méthodes sont nombreuses : demande d'évaluation sur la page du store mobile, récompense pour une action spécifique, etc. Les problèmes peuvent ainsi être rapidement identifiés puis corrigés. La réactivité est aujourd'hui une des clés de la réussite d'une application mobile à succès.

Pour illustrer la différenciation UX entre le Web et le mobile, nous pouvons prendre l'exemple de Facebook. De prime abord, les services rendus par le site Web et l'application mobile semblent identiques. Cependant, lorsque l'on observe un peu plus finement les usages, nous pouvons voir que l'application mobile est centrée sur le flux d'activité et les autres fonctionnalités reléguées au rang d'annexes. Certaines sont même complètement absentes de l'application telle la messagerie instantanée qui fait l'objet d'une application dédiée.

La mobilité au quotidien





Comparaison d'écrans :

en haut, le ***Flat Design iOS*** *et en bas, le* ***Materiel Desing Android.***

Afin d'établir une relation de hiérarchie, les interfaces iOS se servent souvent de l'effet de blur tandis que le Materiel Design d'Android privilégie un effet 3D (utilisation d'ombre, etc.).

Suivre les règles établies par l'écosystème

La notion de Mobile First en UX a grandement amélioré la cohérence de l'écosystème mobile. L'ensemble des applications forme désormais un environnement homogène, notamment grâce au **respect des guidelines des éditeurs des systèmes d'exploitation mobiles** (recommandations officielles visuelles) qui permettent aux utilisateurs de comprendre immédiatement leur fonctionnement. Le **Material Design de Google**[2] ou les **recommandations d'Apple**[3] sont maintenant la norme et se retrouvent dans la majorité des applications. Ces éditeurs aident d'ailleurs les développeurs sur ce point, en mettant à disposition des outils dédiés pour chaque plateforme.

Des plateformes de plus en plus stables mais des applications en perpétuel renouvellement

Le challenge pour les designers, comme pour les développeurs, réside aujourd'hui dans la **cohérence de leurs applications entre les différentes plateformes**. Google a tenté ces dernières années de résoudre le problème de la fragmentation des versions d'Android afin d'offrir un socle cohérent entre devices à l'aide des programmes Android One et des librairies de support. Cependant, cela reste une véritable problématique pour Android où plusieurs OS coexistent. Dans le même mouvement, Apple pousse à la mise à jour (ou à l'obsolescence) des terminaux afin de **minimiser le nombre d'OS à supporter**.

La tendance générale est à la stabilisation : les formats d'affichage se standardisent et les outils de développement (AutoLayout sur iOS, par exemple) permettent de mieux gérer la fragmentation du marché. Enfin, les mises à jour système sont moins radicales, et assurent une meilleure rétrocompatibilité.

Mais là où les OS se stabilisent, les applications mobiles font toujours l'objet d'un renouvellement constant et fréquent.

Mettre en place de nouvelles fonctionnalités ou encore appliquer les guidelines vous garantit une meilleure visibilité : le support des dernières configurations d'écran, des liens universels ou des Instant Apps pour Android encouragent la mise en avant de votre application sur les stores.

L'introduction de nouveaux capteurs ou, tout simplement, l'amélioration des performances des processeurs peut également donner lieu à une mise à jour du logiciel mobile : c'est le cas notamment pour la **mise en place de solutions de Machine Learning** embarquées dans les applications de reconnaissance d'images ou encore pour l'ajout de fonctionnalités de **Réalité Augmentée** dans le cadre d'un magasin de meubles ou d'articles pour la maison.

Côté Android, l'ajout du support des Instant Apps est une autre bonne raison pour renouveler votre application.

2. https://material.io/design/

3. https://developer.apple.com/design/

Instant Apps

La continuité entre Web et mobile et la création d'un flux "sans couture" entre ces deux canaux de consommation de contenus est à la base des Instant Apps, fonctionnalité proposée par Google depuis Google I/O 2016. Le postulat est simple mais révolutionnaire : il s'agit de permettre, à partir d'un lien classique, d'ouvrir non pas une page Web mais une variante réduite de votre application mobile et cela sans installation au préalable.

L'avantage est double : d'un côté l'utilisateur pourra bénéficier d'une **expérience Native** sans passer par le store d'applications ; de l'autre vous augmenterez vos chances de fidélisation car votre application pourra être téléchargée de façon définitive via un raccourci permettant d'installer la version complète de votre logiciel.

Les Instant Apps étant des applications Android en miniature, elles peuvent utiliser la quasi-totalité des capteurs et APIs disponibles : WiFi, Bluetooth, NFC, NDK, etc. Elles sont installables par tout utilisateur ayant au moins Android 5.0 (Lollipop). Les deux principales limitations des Instant Apps sont la **restriction de quelques permissions et surtout leur poids** (le Play Store impose une limite de 4 Mo).

Enfin, sur les stores, votre application représente votre entreprise. Elle se doit de coller à la charte graphique de votre marque : logo, couleurs, polices et tout autre élément qui font référence à votre marque nécessitent une **mise à jour à chaque évolution de votre identité visuelle**.

Pour résumer, les applications mobiles demandent, comme tout logiciel, un soin et une évolution constants. Plusieurs stratégies peuvent être adoptées :

- Pratiquer la politique de la terre brûlée et **rebâtir de zéro une application** à chaque évolution majeure ;
- **Refondre progressivement le logiciel**, en supprimant au fur et à mesure la base de code existante.

Les deux approches présentent risques et opportunités qui dépendent essentiellement de la possibilité de conserver ou non les modules préexistants. Cette étude d'opportunités doit être menée au cas par cas.

Elle est généralement précédée par un choix encore plus structurant : celui de la technologie de développement.

Le choix stratégique de l'approche technologique

Quatre choix se présentent :

- **Technologies natives :** elles permettent de développer une application spécifique pour chaque plateforme (iOS, Android) dans un langage supporté officiellement par l'éditeur de la plateforme (Objective-C ou Swift pour iOS, Java ou Kotlin pour Android) et qui sera installée sur les terminaux.
- **Technologies Web mobiles :** elles consistent à développer une application Web dédiée au mobile et utilisée depuis les navigateurs du terminal, tels que Safari ou Chrome. Les Progressive Web Apps **(PWA)** en sont l'exemple le plus avancé.
- **Technologies hybrides :** elles permettent d'encapsuler une application Web dans une application « coquille » native afin de faciliter l'accès aux fonctionnalités du terminal du dispositif (tel que l'appareil photo) à partir de code JavaScript et de proposer son installation sur les devices.
- **Technologies natives cross-platform :** elles se servent d'un moteur de rendu natif pour garantir performance et fluidité lors de l'usage de l'application. Votre logiciel sera développé à l'aide d'un framework autre que ceux officiellement supportés par les éditeurs d'OS mobiles. Les exemples les plus connus sont Xamarin, AppCelerator, React Native, NativeScript ou Flutter.

Application native

Si vous avez besoin d'une interface graphique avancée, avec une haute qualité visuelle ou des animations Pixel Perfect, de hautes performances ou d'un accès à toutes les fonctionnalités du téléphone alors faites le choix du natif. **C'est le choix de la qualité optimale**. Votre application sera disponible sur les stores ce qui lui assurera une bonne visibilité. Enfin, vous pourrez prévoir des **cas d'utilisation hors ligne** et donc un accès constant aux informations.

Cependant, les **coûts d'un projet natif seront plus élevés** si votre cible contient plusieurs plateformes. En effet, vous aurez à réécrire l'intégralité de la base de code pour chacune d'elles.

De même, si vous prévoyez une application pour mobile et tablette, il faudra recréer l'interface visuelle au complet même si une grande partie de la base de code sera identique.

Web Mobile et PWA

Si votre objectif est de **proposer rapidement et à moindre coût** une application disponible sur toutes les plateformes, optez pour un site Web mobile. Vous pourrez mettre en place un design responsive qui assurera un bon fonctionnement quel que soit le terminal.

Cependant, l'application ne sera **pas disponible sur les stores** et sera uniquement accessible depuis un navigateur Web. Elle sera donc moins visible mais plus facilement mise à jour car non soumise aux cycles de validation d'Apple et Google. Enfin, la **fluidité de l'interface sera moins bonne ce qui appauvrira l'expérience utilisateur**.

Grâce à un usage attentif des nouvelles APIs Service Worker, elle pourra fonctionner en mode hors ligne, présenter des notifications push, réaliser des tâches en arrière plan ou encore être installée de façon similaire à une application native.

Traditionnellement "oubliées" par Apple, les technologies PWA commencent à gagner quelques formes de support.

Pour aller plus loin sur les Progressive Web Apps, consultez notre dernier TechTrends Front[4].

Accelerated Mobile Pages et Instant Articles

Même si elles ne sont pas directement liées à la notion de PWA, Google et Facebook proposent aujourd'hui des librairies Web permettant d'accélérer (jusqu'à 4 fois) les temps de chargement des pages Web depuis un navigateur mobile : il s'agit des **frameworks AMP** (Accelerated Mobile Pages) et Instant Articles.

Ces solutions tirent profit des solutions de caching des navigateurs modernes pour **réduire considérablement le contenu chargé à la demande**. Les contraintes imposées sont par contre majeures : l'impossibilité de charger un script JavaScript custom ou de pratiquer du tracking en sont de bons exemples. AMP et Instant Articles ne s'appliqueront finalement correctement qu'aux **contenus statiques de votre site**.

Tout comme pour les PWA, les pages accélérées de Google et Facebook sont des technologies exclusivement orientées Web mais qui peuvent améliorer l'expérience de certains de vos utilisateurs mobiles.

Application hybride

Si votre application répond à un **besoin simple**, comme de la saisie de formulaire ou de l'affichage de listes sans recherche et ne nécessite pas une forte réactivité, alors une application hybride sera satisfaisante. Vous pourrez la mettre à disposition sur les stores et utiliser certaines fonctionnalités natives comme les Push Notifications.

Cependant, l'expérience utilisateur sera dégradée puisqu'elle **se basera sur des composants Web et non natifs**. Si le développement d'un design responsive commun est possible, des développements spécifiques à chaque plateforme seront tout de même nécessaires afin que l'application soit affichée correctement. De même, chaque évolution devra prendre en compte les différents OS, ce qui complexifiera grandement le code et le rendra moins maintenable. Enfin, la gestion des cas aux limites est plus complexe car elle passe par un framework tiers qui doit les interpréter. Tous ces critères font que le développement ne sera pas nécessairement moins coûteux.

4. Le TechTrends Front est disponible en téléchargement sur https://www.publicissapient.fr/services/engineering

Application native cross-platform

En ce qui concerne les applications natives cross-plateform, il nous semble, à l'heure où nous écrivons ce TechTrends, que la plupart des technologies (Xamarin, Appcelerator, React Native, Flutter) ne sont pas encore assez mûres. L'expérience utilisateur est rarement adaptée au terminal et des **développements spécifiques à chaque plateforme** sont nécessaires. Vos coûts augmenteront donc sans nécessairement vous donner une finition optimale.

Comment choisir ?

Les critères à prendre en compte dans votre choix sont nombreux et dépendent de vos besoins. Voici les paramètres qui nous semblent primordiaux :

- **Multi-Device** : capacité à développer une seule application qui sera correctement affichée quelles que soient la taille et la résolution de l'écran (smartphone vs tablette par exemple).
- **Réactivité** : capacité à réagir rapidement et de façon fluide aux différentes interactions afin d'optimiser l'expérience utilisateur.
- **Résilience** : capacité de l'application à fonctionner en cas de panne des autres systèmes dont elle dépend.
- **Coût** : ensemble des coûts de développement et de maintenance de l'application.
- **Maintenabilité & pérennité** : capacité à faire évoluer facilement, corriger les bugs de l'application et garder une base de code saine dans le temps.
- **Accessibilité** : présence sur le store de la plateforme.

Chez Xebia, nous prenons très souvent le parti du natif car il représente pour nous le choix de la qualité. La différence avec les autres technologies peut être, selon la complexité de vos besoins métier, très marquée. Nous traiterons donc dans la suite de ce TechTrends de choix natifs, la plus grande partie de nos préconisations restant applicables à toutes les technologies mobiles confondues.

Besoin / Technologie	Native	Site Web mobile / PWA	Cross-platform	Native Cross-platform
Multi-Device	⚠	✓	⚠	✗
Réactivité	✓	✗	✗	✓
Résilience	✓	⚠	⚠	✓
Coût	✗	✓	⚠	⚠
Maintenabilité & pérennité	✓	✓	⚠	⚠
Accessibilité	✓	✗	✓	✓

Zoom sur les frameworks Cross-Platform natifs

React Native : React + JavaScript pour Android et iOS

React Native, disponible en open source sur GitHub, est développé par Facebook (avec de nombreuses contributions de la part de la communauté) depuis 2013. Le framework **s'inspire fortement de React** et d'une architecture orientée composant : la promesse est de pouvoir développer des applications mobiles riches pour iOS et Android à l'aide des mêmes technologies, mutualisant ainsi une grande partie du code y compris celui dédié à la couche de présentation. Le paradigme de développement étant dérivé du framework Web React, les développeurs Front-End pourront rapidement devenir opérationnels ou, a minima, comprendre facilement les sources d'une application React Native.

S'appuyant sur des composants OEM *(Original Equipment Manufacturer)*, les applications React Native permettent d'utiliser les éléments natifs du système cible avec une convention uniforme entre les différentes plateformes.

Le code métier, en JavaScript, est exécuté directement sur les terminaux et la communication avec le système se fait via un *bridge*. Ces opérations sont exécutées dans un thread dédié, ce qui permet de ne pas bloquer le thread principal de l'application.

L'écueil principal des applications React Native est la performance. Le bridge et le moteur JavaScript ne permettent pas d'arriver au niveau des applications natives même si, en y dédiant du temps, il est possible de s'en approcher. À noter également qu'il est toujours nécessaire à un moment ou à un autre de faire du code spécifique natif (iOS ou Android) si le composant React Native correspondant n'est pas disponible.

Flutter : Component + Dart 2

Flutter, également disponible en open source, est développé par Google depuis 2017. Le framework est **orienté composant et réactivité**, comme React Native. Les composants Flutter peuvent être facilement étendus, ce qui permet aux développeurs **d'ajouter un comportement personnalisé aux modules** déjà codés par la communauté *(Material Design ou Cupertino)*.

À la différence de React Native, la promesse de Flutter est de réaliser une **application *Brand Oriented* présentant le même design sur iOS et Android :** par conséquent le framework ne propose pas de composants *OEM* du système cible. À la place, Flutter utilise un moteur 2D, *Skia*, racheté en 2005 par Google, qui gère le rendu des composants de l'application.

Le code Dart des applications Flutter est compilé vers les instructions natives de la plateforme cible (iOS ou Android) : il n'y a donc pas d'interpréteur ou de *bridge* comme sur React Native.

Le framework souffre d'une relative jeunesse et donc d'un manque de ressources. Comme pour React Native, il faudra parfois écrire du code natif pour interagir avec le système (réseau, caméra, etc.) même si la création de composant est grandement facilitée grâce au moteur 2D.

Un SI au service de ses applications nomades

Une fois la technologie de votre application mobile choisie, il convient de s'interroger sur les adaptations nécessaires de votre Système d'Information. En effet, le paradigme Mobile First a aujourd'hui dépassé les frontières de l'UX et influence directement l'architecture et l'infrastructure d'un SI. Une application mobile va avoir des besoins spécifiques qui n'ont pas toujours été prévus lors de la création du SI.

Parmi les aspects les plus importants à prendre en considération, **les performances, la sécurité et la gestion des mises à jour nécessitent une attention particulière**.

Un applicatif mobile requiert des services optimisés et performants. En effet, une application distribuée publiquement est une véritable vitrine pour votre entreprise. Les performances et la disponibilité du SI qui la supporte vont donc être primordiaux : il faudra ainsi **estimer la charge et correctement dimensionner vos plateformes** (comme pour un site Web ou une application Desktop). De même, une application mobile doit pouvoir fonctionner dans des conditions de réseau dégradées : les services devront répondre rapidement et échanger la juste quantité de données. Cette optimisation des échanges est vitale pour le mobile. Il faut réellement prendre en considération les conditions réseaux lors du développement des services : si trop de données sont renvoyées cela utilisera inutilement la bande passante et alourdira les traitements sur l'application.

La sécurité est également un élément à considérer. Une application peut se retrouver installée sur une flotte de terminaux non "maîtrisés". La politique de prévention des risques doit donc être mise à jour régulièrement afin de **faire face à de potentielles attaques**. La gestion des sessions, des certificats SSL, la robustesse des validations de données côté serveur, l'injection malicieuse côté client sont autant d'exemples de failles de sécurité que votre SI doit prendre en considération avant de distribuer un applicatif mobile[5].

Enfin, un aspect fondamental mais souvent sous-estimé du cycle de vie d'une application mobile est sa mise à jour. Nouvelles fonctionnalités, changements structurants introduits lors des mises à jour d'OS ou simples corrections de bugs nécessitent de la republier sur les stores. Si une modification de vos services SI devient nécessaire, par exemple pour un problème de performance ou de sécurité, il faut **prévoir que toutes les applications ne seront pas mises à jour simultanément**. Vous aurez donc soit à garantir une compatibilité ascendante via une gestion intelligente des contraintes sur les contrats de services au cours du temps, soit à gérer plusieurs versions de vos services en même temps pour offrir une continuité d'utilisation. Une dernière solution consiste à implémenter un système de mises à jour forcées de votre applicatif.

5. https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks (https://bit.ly/1pyxK6L)

Take away

Comprendre

Réaliser

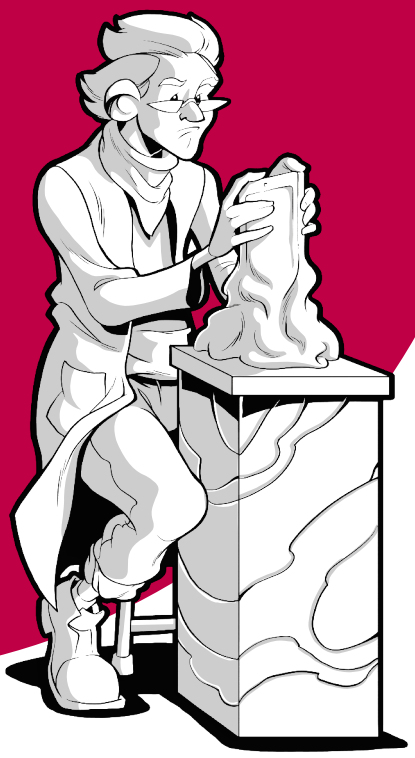
Réaliser son application spécialement pour le mobile que ce soit au niveau des fonctionnalités ou de l'UX. Ne pas réaliser un simple portage.

Choisir

Choisir les technologies de développement en analysant finement tous les paramètres et pas seulement le coût.

Adapter

Adapter son SI afin de servir au mieux les usages "nomades".



Façonner

Une application mobile répond très souvent à un besoin simple et facilement exprimable.

Cette simplicité ne doit cependant pas masquer la complexité de sa réalisation qui doit être abordée en s'appuyant sur des compétences, des méthodologies, des technologies et des outils choisis avec soin.

Une application mobile s'inscrit aujourd'hui dans un contexte bien plus large que la simple exécution de code sur un terminal portable : comme tout logiciel moderne, elle est en effet le résultat d'un processus de conception et développement qui doit tenir compte de nombreux aspects tels que les **méthodologies de suivi du projet, l'intégration avec le SI, la stratégie de tests, le processus d'intégration continue, le cycle de déploiement et la maintenance évolutive**.

Nous ne passerons pas ici en revue l'ensemble des bonnes pratiques de conception logicielle, mais insisterons sur les spécificités liées aux applications mobiles.

Gérer son projet : agile, évidemment

Lors de la création d'un produit mobile, il est souvent nécessaire d'adresser en même temps plusieurs plateformes (iOS et Android par exemple). Nous avons dégagé de nos expériences un ensemble de bonnes pratiques permettant de gérer au mieux cette spécificité dans un cadre agile :

- **Une organisation inspirée des Feature Teams** : il est essentiel de rassembler au sein de la même équipe les développeurs de chaque plateforme, les développeurs du back-end et les designers/ergonomes, accompagnés par un Product Owner unique et, si possible, un équipier dédié à la recette (une bonne pratique que l'on ne rencontre que trop peu souvent). Au delà des gains évidents de proximité et d'échange, cette organisation permet de **mettre à profit la pluri-disciplinarité des experts iOS et Android et de capitaliser sur les problématiques back** rencontrées et les solutions apportées ainsi que sur la compréhension globale du fonctionnel.

- **Un Backlog de fonctionnalités dédoublé :** la complexité d'une Story mobile varie d'une plateforme à l'autre. Il est donc nécessaire de créer des Stories mobiles pour chaque plateforme afin de les estimer séparément et de les laisser suivre leurs Cycle Times propres. Cette approche offre une meilleure prédictibilité.

- **Des cycles d'itérations synchronisés :** lors d'une itération, il est intéressant, si cela est possible, de développer un ensemble de fonctionnalités quasi identiques sur la totalité des plateformes. En effet, lors des différentes cérémonies, les développeurs pourront aborder des problématiques communes et échanger. De même, si un bug est rencontré, l'analyse en sera facilitée (par exemple, s'il est présent sur tous les OS, il y a de fortes chances qu'il provienne des services back-end plutôt que des applicatifs mobiles).

- **Une planification des releases à géométrie variable** : nous avons constaté que, bien souvent, les développements ne peuvent pas être réalisés à la même vitesse sur tous les OS. Un choix doit donc être fait quant à la planification des releases : attendre que toutes les applications couvrent le même scope fonctionnel ou accepter une désynchronisation des services offerts sur les différents OS. Dans le cas d'une première release ou d'une mise à jour majeure, il est courant d'attendre. En revanche, lorsqu'il s'agit de mises à jour mineures, le décalage fonctionnel est acceptable. La synchronisation parfaite des fonctionnalités est, à notre avis, utopique.

- **Des releases à chaque sprint** : une fois la release majeure publiée, toute mise à jour ultérieure peut grandement bénéficier d'un cycle de publication fréquent. Une livraison à la fin de chaque sprint permet effectivement de recueillir rapidement le feedback des utilisateurs ainsi que de détecter immédiatement la survenance de régressions.

L'orientation agile évoquée ici doit être **appuyée par une approche radicalement qualitative de la réalisation logicielle.**

Choisir son langage de développement

iOS : Swift

Swift est le successeur désigné d'Objective-C. Sa syntaxe, plus concise, simplifie la lecture et rend le code plus intelligible. C'est un langage fortement typé, qui lèvera donc des erreurs dès la compilation et permettra ainsi de mieux sécuriser l'exécution de la logique applicative. Il facilite également la mise en place de paradigmes de programmation fonctionnelle.

D'autres nouveautés font de **Swift un langage plus moderne et expressif** grâce à des fonctionnalités que vos équipes sont ravies de pouvoir utiliser : optionals, generics, extensions de protocoles, pattern matching et bien d'autres. Swift est aussi plus rapide à l'exécution qu'Objective-C dans de nombreux cas de figure[6].

Depuis la toute première version du langage, nous avons mis en production plusieurs applications entièrement réalisées en Swift. La stabilité et la concision sont probantes : les **erreurs de développement sont plus facilement anticipables et le taux de crash diminue grandement**. La migration entre versions majeures du langage se fait de plus en plus simplement à chaque release.

Objective-C est-il donc en fin de vie ? Pas nécessairement car, à date, il reste fonctionnel et toujours au cœur, pour des raisons historiques, de la plupart des applicatifs iOS. De plus, toute nouvelle API proposée par Apple, de ARKit à Core ML, est rendue systématiquement accessible aux deux langages de programmation.

Néanmoins, nous recommandons fortement Swift pour le développement d'une nouvelle application.

Android : Kotlin

Afin de dépasser certaines limitations de Java lors du développement de leurs IDE IntelliJ et Android Studio, la société JetBrains a créé en 2011 un nouveau langage JVM : Kotlin. Ce dernier présente de nombreux avantages pour le développement Android, à commencer par son **intégration naturelle dans les IDE susnommés**. Reposant sur des concepts modernes, Kotlin se veut très intuitif pour des développeurs Java.

Un autre véritable atout de Kotlin concerne la robustesse du langage lors de son exécution : par exemple, les variables ne peuvent être *null*, ce qui permet de ne pas avoir à vérifier l'état d'une variable avant de l'utiliser. L'erreur la plus fréquente en Java, la fameuse *"NullPointerException"* est ainsi éliminée. Les conversions entre types sont aussi mieux gérées lors de la compilation.

Kotlin **encourage la programmation fonctionnelle et améliore la lisibilité du code** grâce à une syntaxe plus concise. Son utilisation apporte une vraie plus-value, comme par exemple lors de l'utilisation de frameworks de "Reactive Programming" tel que RxJava. L'écriture du code est alors simplifiée par l'utilisation de Lambda et permet de supprimer le code superflu afin d'en augmenter la lisibilité.

Il est possible de **faire cohabiter au sein d'une même application du code Java et du code Kotlin**. Cette fonctionnalité étant prévue nativement par le langage, il est très facile d'appeler du code Kotlin depuis Java et inversement.

Kotlin n'est plus aujourd'hui un langage de niche et son adoption va grandissante. Il est désormais supporté par Google dans le cadre du développement Android tandis que Spring et Gradle offrent une API compatible avec le langage de JetBrains.

6. https://developer.apple.com/swift/blog/?id=27 (https://apple.co/23LqwAp)

Développer son back-end

Aujourd'hui, la distinction entre équipes mobiles et back-end a tendance à s'atténuer, voire à complètement disparaître.

Applications "Full-Stack" en Kotlin ou Swift

L'avènement de Swift et Kotlin modifie fondamentalement le paysage mobile : il est aujourd'hui possible de construire des applications Full-Stack qui partagent **un seul et unique langage** dans leurs déclinaisons back-end et front-end.

L'avantage est double : chaque développeur peut intervenir efficacement à la fois sur les codes client et serveur ; il est possible de **réutiliser certaines portions de code** (les objets métier, par exemple).

D'un point de vue technique, Swift, par l'intermédiaire de frameworks tels que Kitura (porté par IBM) ou Vapor, est une alternative viable pour le développement d'applications back-end. Parmi ses principaux atouts citons **sa rapidité, une faible consommation de ressources système, la prédictibilité de son empreinte mémoire et un typage fort**.

Kotlin, bien inscrit dans le monde Java, été pensé d'emblée comme un langage back-end. Sa compatibilité avec Spring est évidemment un atout majeur.

De manière plus étonnante, le monde iOS commence à s'intéresser à Kotlin. Grâce à Kotlin/Native, un compilateur capable de générer des binaires natifs pour les devices de la Pomme, la mutualisation de code Kotlin entre applications construites pour des systèmes d'exploitation différents est envisageable dans un futur proche. Le doux rêve de voir enfin une réconciliation entre iOS et Android est peut-être à portée de main.

Back-end as a Service

Grâce à l'introduction, il y a quelques années, des premiers Back-end as a Service (BaaS), la difficulté de développer un back-end diminue. Cette approche s'inscrit dans le modèle d'architecture serverless dans lequel un fournisseur de services cloud gère dynamiquement l'allocation des ressources. Les complexités liées à la gestion et à la montée en charge des serveurs sont ainsi totalement cachées au développeur.

Les offres de Back-end as a Service permettent aux développeurs **l'intégration à l'aide d'APIs haut niveau de services pré-packagés** tels que la gestion des utilisateurs, la synchronisation des données ou encore les notifications push. Les services proposés se font de plus en plus nombreux tout comme les fournisseurs les proposant : citons Firebase de Google ou encore AWS Mobile Hub. N'oublions pas non plus CloudKit, proposé par Apple, qui a le défaut de ne supporter que les applications iOS.

Dans ce contexte, une nouvelle catégorie de services a vu le jour : il s'agit de **Function as a Service (FaaS),** véritables micro-programmes dont l'exécution est déclenchée à la requête explicite de l'application mobile ou à la vérification de conditions définies par le développeur.

Échanger les données

Les terminaux nomades étant, dans la plupart des cas d'usage, de simples interfaces de communication entre l'utilisateur final et le SI, une attention particulière doit être portée aux interactions entre client et serveur.

Un terminal actif : le mode Pull

Comme beaucoup de composants front-end, le mobile consomme la plupart du temps des services fournissant du JSON. Deux bonnes pratiques sont à appliquer systématiquement en mode pull pour limiter au maximum le volume de données échangées :

- **Ne faire qu'un seul appel réseau** pour récupérer la totalité des données d'un écran ;
- **Ne renvoyer que les informations strictement nécessaires**.

Ceci améliorera la dynamique de votre application et permettra d'optimiser plus facilement les performances globales de votre chaîne applicative.

Comme pour les applications Web, le langage de requêtage **GraphQL**, conçu par Facebook, a le vent en poupe. GraphQL permet la création de requêtes dont la structure de retour est définie par le client. Cette approche confère donc au client le pouvoir de sélectionner uniquement les données qui lui sont nécessaires. Il en résulte une diminution significative du payload de réponse.

Un terminal passif : le mode Push

Dans certains cas d'usage, il est de la responsabilité du serveur de notifier ses clients mobiles.

Pour ce faire, deux technologies cohabitent : **les notifications push et les sockets**.

Les applications mobiles utilisent depuis toujours massivement les notifications push pour délivrer au plus tôt une information à l'utilisateur, sans pour autant garantir du temps réel. Ces notifications nécessitent des développements spécifiques côté serveur pour s'interfacer à la fois avec **APNS** (Apple Push Notification Service) et **GCM** (Google Cloud Messaging).

Même si les services de push standards d'Apple et de Google sont gratuits, l'intégration de leurs APIs dans votre SI nécessite des implémentations spécifiques et laborieuses. Afin de pallier ces difficultés, des **solutions de type SaaS** ont vu le jour. Ces solutions parfois payantes apportent une qualité de service accrue et des statistiques d'usage précises. Certaines offres, comme Amazon PinPoint ou Salesforce, orientées campagnes marketing permettent aussi de pousser des messages promotionnels en adressant plusieurs autres canaux de communication, tels que le courrier électronique ou les SMS.

En parallèle de ce système dorénavant « classique », les **WebSockets**[7] permettent d'établir un canal de communication bidirectionnel et performant entre application et SI. Par exemple, les jeux multi-joueurs ou bien les dashboards utilisent ce type de notifications afin **d'assurer une cohérence et une réactivité maximale** entre les différents utilisateurs. Parmi les librairies permettant la mise en place de WebSockets, citons **OkHttp pour Android et SocketRocket pour iOS**.

Notons l'approche originale de **Firebase** qui permet, par le biais d'une même interface de programmation, de mettre en place à la fois WebSocket et Notifications Push. La complexité des deux implémentations est abstraite par l'API fournie par le framework.

7. https://fr.wikipedia.org/wiki/WebSocket (https://bit.ly/2kxwlT3)

Tester les applications mobiles

Les tests sont d'autant plus vitaux dans le monde mobile que le cycle de déploiement sur les stores peut être long (de quelques heures à deux semaines en fonction de la cible et de la richesse de l'application). Si l'application livrée présente un bug, il faudra de nouveau attendre avant de pouvoir fournir un correctif aux utilisateurs.

Dans ces conditions, il est primordial de mettre en œuvre une stratégie de tests efficiente dès le début du projet et de la maintenir tout au long de la vie du produit. C'est elle qui garantit que le code est résilient aux aléas d'une vie en production. Les applications mobiles sont en effet soumises à des parcours d'usage complexes qui provoquent de nombreux cas aux limites : réception d'événements système, perte de réseau, mise en tâche de fond de l'application, etc.

Afin de s'assurer du maintien de cette stratégie de tests, **il faut la monitorer**. Le suivi des métriques relatives aux tests permet de prendre les actions nécessaires pour garantir la maintenabilité et la pérennité d'une application.

Deux spécificités font de **la mise en place de tests fonctionnels sur mobile une étape difficile mais indispensable** :

- Les applications mobiles sont soumises à une multitude d'interactions : gestures tactiles *(tap, swipe, pinch, etc.)*, événements système, boutons physiques, etc. Ces interactions devront être implémentées dans l'outil de test et chaque comportement utilisateur devra y être décrit précisément.
- Le cycle d'exécution des différentes tâches d'une application sur mobile est complexe car multithreadé (les appels réseaux et les traitements longs sont exécutés dans des threads différents de celui qui exécute la vue). Les tests doivent prendre en compte cet aspect et **gérer finement la synchronisation des threads**. Comme il est nécessaire d'attendre la fin des exécutions sur tous les threads avant de passer à l'étape suivante, il est possible qu'un test soit en échec non pas à cause d'un bug mais à cause d'un problème externe (réseau médiocre ou CPU occupé par exemple).

Malgré toutes ces difficultés, ces tests sont déterminants dans le suivi des parcours critiques. Ils garantiront la résilience de l'application et l'absence de régressions sur votre cœur de métier.

Vos équipiers qualité seront également déchargés de l'exécution manuelle des tests les plus complexes à reproduire et pourront se concentrer sur des scénarii à plus haute valeur ajoutée. En effet, les tests d'intégration ne dispensent pas des tests manuels. La meilleure option pour les mises à jour importantes est de mettre à disposition une version bêta auprès d'un groupe choisi d'utilisateurs pour valider les nouvelles fonctionnalités. Pour les autres versions, il faut a minima que le Product Owner puisse disposer d'une version à tout moment afin de valider les tâches terminées par l'équipe.

Le côté primordial et répétitif de ces tâches a fait naître un besoin **d'automatiser ces processus**, et c'est ainsi que sont apparus les **systèmes d'intégration continue**.

Intégration Continue (CI)

Se doter d'une plateforme d'intégration continue capable de construire les applications Android et iOS n'est pas trivial. Il faut distinguer deux types d'outils : hébergés (ou On Premise) et à la demande (ou SaaS).

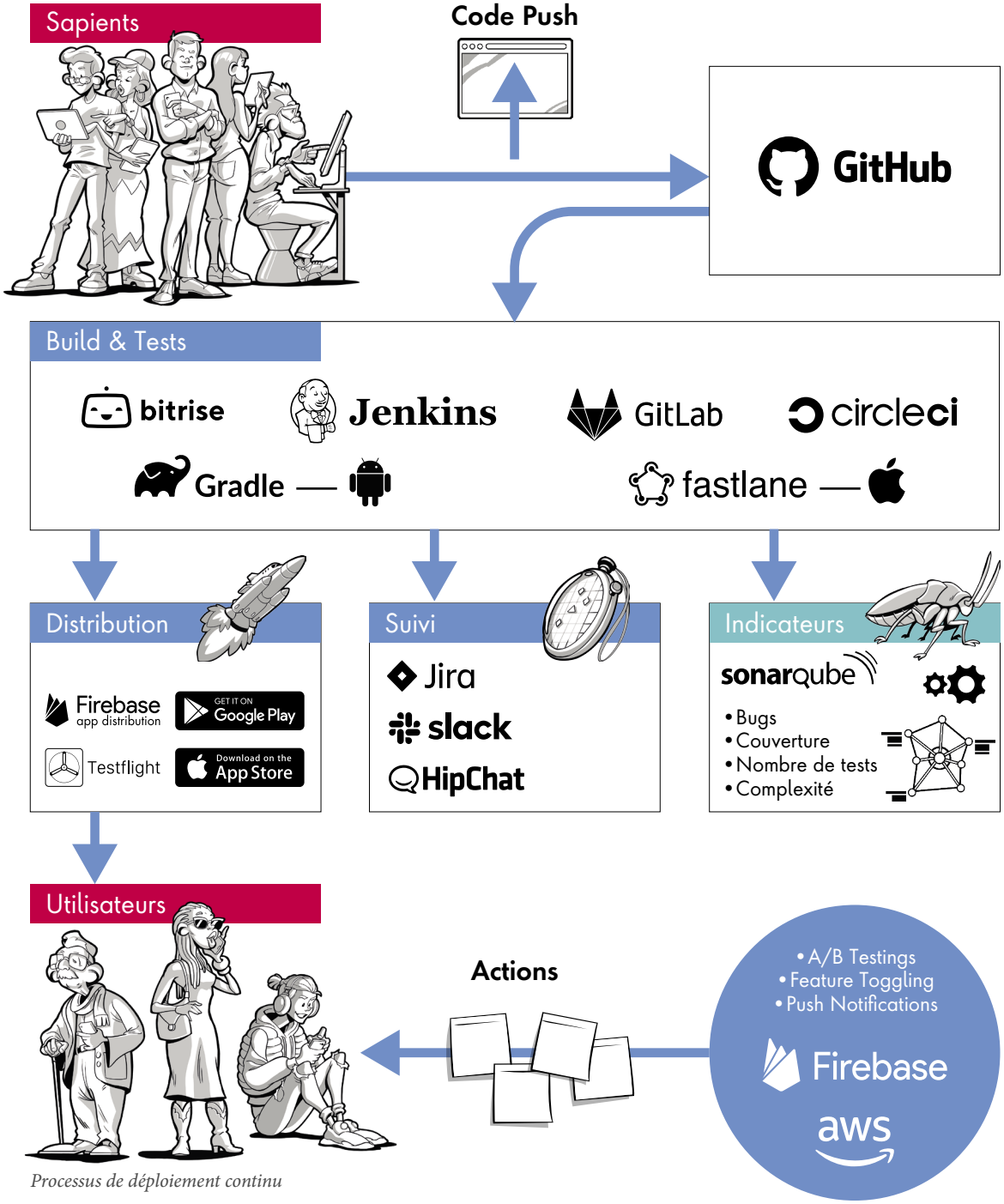
Traditionnellement peu robustes et peu fiables, les plateformes d'Intégration Continue dédiées à la mobilité ont récemment connu une évolution sans précédent.

Si d'un côté les **solutions On Premise telles que Jenkins ou, plus récemment, GitLab CI** répondent toujours aux exigences fondamentales d'automatisation, elles impliquent un investissement constant de la part des équipes afin de maintenir le matériel et les outils à jour. C'est pour cela que, depuis 2015, de nouveaux services à la demande s'imposent au sein de la communauté des développeurs mobiles.

Il s'agit notamment de solutions telles que **CircleCI ou Bitrise** qui mettent à disposition des développeurs **des interfaces spécifiques aux besoins de la mobilité** (signature et testing avant tout) et simplifient grandement le processus de build. Elles proposent généralement une offre gratuite limitée à quelques minutes de temps de build. Les offres payantes démarrent à 50$/mois et prennent en charge iOS et Android. Le système est automatiquement géré, mis à jour et maintenu par le fournisseur. Au-delà des aspects coûts, les inconvénients majeurs de ces solutions par rapport aux alternatives hébergées sont leur manque de personnalisation et leurs performances parfois limitées.

Sur iOS, Xcode Server est une alternative toujours fonctionnelle mais la récente acquisition par Apple du service de build SaaS Buddybuild laisse entrevoir une nouvelle promesse.

Déploier



Processus de déploiement continu

Distribuer votre application à une population choisie

Pour mettre à disposition de vos utilisateurs les différentes versions de votre application, il existe des **solutions de MAM (Mobile Application Management)**. Elles permettent notamment la livraison des applications, leur téléchargement et leurs mises à jour. Très souvent, ces solutions permettent également de suivre les versions téléchargées par les utilisateurs et de remonter des statistiques d'usage et les plantages survenus. Grâce à un système de gestion de droits, vous pouvez facilement vous assurer de ne livrer vos versions intermédiaires qu'à un groupe défini de personnes. Enfin, ces outils assurent le suivi des versions d'une application et permettent ainsi de télécharger une ancienne version si besoin.

Parmi les outils du marché, nous pouvons citer **App Distribution de Firebase et HockeyApp** qui permettent de gérer les applications iOS et Android. TestFlight, racheté par Apple, est aujourd'hui directement intégré à iTunes Connect et permet de facilement mettre à disposition une bêta de son application (sur iOS uniquement). De même, Google met à disposition Android Alpha / Beta testing directement sur le Play Store qui permet en un clic de pousser en production son application à une population choisie.

En entreprise, si vous souhaitez contrôler finement les accès à votre parc d'applications, un outil MAM n'est pas suffisant. Vous devrez vous appuyer **sur un service de MDM (Mobile Device Management)** qui enrichira la gestion de vos applications avec diverses options. Vous pourrez par exemple gérer des flottes de terminaux en entreprise en reliant un mobile à son utilisateur et en définissant quelles applications doivent y être installées. Ces outils facilitent également la gestion des profils de configuration en paramétrant à distance les proxies, VPN ou autres configurations réseau spécifiques.

L'aspect sécuritaire est également un atout fort de ces solutions car elles permettent l'effacement total des données sur un terminal depuis l'outil, le chiffrage des données, la protection par mot de passe ou encore la gestion des échanges de fichiers entre les applications installées.

Les informations du terminal telles que la position, la batterie ou la qualité du réseau sont remontées en temps réel. La principale limitation de ces outils vient du support des différents OS. Il est souvent nécessaire d'attendre quelques temps avant que les dernières versions soient supportées.

Grâce aux outils MAM et MDM, vous pourrez facilement distribuer votre application. Afin de vous assurer de la qualité de vos développements, profitez-en pour lancer au plus tôt des versions bêta à un groupe choisi d'utilisateurs.

Distribuer votre application au grand public

Si votre application est à destination du grand public, vous allez devoir la mettre à disposition sur les stores d'Apple et de Google. Voici quelques conseils afin de vous faire repérer parmi la kyrielle d'applications proposées.

Tout d'abord, **pensez votre page store comme une véritable vitrine** pour votre application. C'est le point d'entrée pour tous les nouveaux utilisateurs. Elle se doit donc d'être particulièrement soignée et de mettre en avant les différentes fonctionnalités de l'application en suscitant l'envie. Elle doit également être facilement accessible. Choisissez donc bien les mots-clés qui la feront remonter dans les résultats de recherche.

Ensuite, il vous faut un **maximum de commentaires**. Des études ont montré que près de 3 utilisateurs sur 4 regardent la notation d'une application avant de choisir de la télécharger. Engagez vos utilisateurs au sein de votre applicatif afin de les pousser à le noter sur le store. N'hésitez pas à répondre aux commentaires de vos utilisateurs. Cela montre que vous prenez en compte leurs retours et permet parfois de compenser une note faible.

Monitorer

Une fois l'application déployée, il est primordial de la monitorer afin de suivre son évolution au cours des mois et des années.

Ce monitoring concerne deux classes d'informations : les éléments techniques du logiciel et les mesures liées à l'usage de l'application.

Métriques techniques

L'analyse de ces métriques a pour but de connaître, suivre et réduire l'inévitable dette technique qui s'accumule tout au long d'un projet : il s'agit d'indicateurs tels que la couverture de code ou la complexité cyclomatique.

Les IDE mobiles actuels permettent de réaliser aisément le **profiling de votre logiciel** et donc d'identifier de potentiels problèmes tels que la surconsommation de batterie, les fuites de mémoire ou des ralentissements de l'interface utilisateur.

Métriques d'usage

Le suivi des usages doit vous aider dans vos prises de décisions. En analysant les parcours utilisateur, vous serez mieux à même de répondre efficacement à leurs attentes. Les outils actuellement disponibles sur le marché vous permettront de suivre l'adoption, le parc de terminaux, les parcours ainsi que le taux de crash et d'autres anomalies. Ces indicateurs sont de bons candidats au **management visuel**, dans vos open spaces, afin que toute l'équipe, du métier à la production, soit sensibilisée aux usages effectifs de votre application et aux difficultés potentiellement rencontrées par vos utilisateurs.

La boîte à outils du développeur iOS/Android

Les mondes iOS et Android ont la chance d'avoir des **communautés de développeurs actifs et prolifiques**. En plus du support d'Apple et de Google, de nombreuses bibliothèques sont développées par des tiers pour simplifier certains processus courants dans une application mobile, tels que les couches réseaux ou les tests. Au début d'un projet se pose toujours la question des outils à utiliser. Nous avons rassemblé ici ceux qui nous semblent les plus pertinents à l'heure où nous écrivons ce TechTrends.

Android

iOS

IDE

Android Studio, édité par JetBrains, est l'outil officiel de développement Android. Il inclut de nombreuses fonctionnalités comme la construction des écrans en glissant les composants directement dans l'IDE ou l'intégration du gestionnaire de build (ce qui permet de lancer la compilation, les tests et le déploiement directement depuis l'IDE).

Android Studio permet également de gérer les différentes versions du SDK, les terminaux de tests, les déploiements, la construction des binaires et même des émulateurs performants.

Il inclut la mesure des performances (utilisation CPU et mémoire) en temps réel et le rendu des écrans sur plusieurs appareils / versions / langues en même temps.

Android Studio intègre également le support du langage Kotlin.

Xcode est la solution de développement officielle, créée et maintenue par Apple. Elle intègre de nombreux outils très pratiques lors de la création d'une application : gestion des interfaces via les storyboards, gestion des SDKs et des versions du simulateur, outils de mesure des performances et beaucoup d'autres. Il est également l'IDE le plus fiable en ce qui concerne le support de Swift.

Cependant, d'autres outils ont vu le jour et méritent d'être mis en avant. Tout d'abord, **AppCode** qui propose une meilleure autocomplétion et permet de facilement refactorer ses sources. Citons aussi **Reveal.app**, un outil avancé, pour débugguer les vues de l'application.

Build et gestion des dépendances

La gestion des dépendances pour Android se réalise à l'aide de **Gradle**, outil officiel de build très bien intégré dans les projets avec Android Studio.

Gradle repose sur des fichiers de configuration écrits en Groovy ou en Kotlin qui listent l'ensemble des librairies et les versions souhaitées.

Côté iOS, il n'existe pas d'outil officiel pour gérer les dépendances. Cependant, un standard s'est imposé de fait ces dernières années : **CocoaPods**. Beaucoup de librairies sont aujourd'hui disponibles avec l'outil. Il génère un fichier workspace qui inclut le projet initial et un projet contenant l'ensemble des bibliothèques. Celles-ci seront ainsi compilées en même temps que le projet.

Il existe également un autre système de gestion des dépendances appelé **Carthage**. Celui-ci va pré-compiler les dépendances et générer des fichiers frameworks à intégrer manuellement au projet.

En outre, un gestionnaire de dépendances officiel, développé initialement par Apple dans le cadre de l'initiative Swift, a vu le jour : il s'agit de **SPM (Swift Package Manager)**. Il prend également en charge les plateformes Linux.

Architecture


Construire une application mobile demande de solides compétences en architecture logicielle. Il existe des modèles éprouvés sur lesquels le développeur peut capitaliser : **Model View Presenter (MVP), Model View ViewModel (MVVM), View Interactor Presenter Entity Routing (VIPER) et Clean Architecture**. Dans toutes ces architectures, la clé réside dans le découplage des différents éléments de l'application.

MVP et MVVM proposent un découpage en trois parties : la vue qui récupère les interactions de l'utilisateur, le présentateur qui coordonne les actions de l'utilisateur et l'interaction avec les données et le modèle qui récupère ces données. La différence entre MVP et MVVM réside dans le fait d'adhérer à un paradigme d'interfaces (MVP) ou de programmation réactive (MVVM). Ces deux architectures suffisent pour découpler le code et ainsi favoriser les tests, le travail en équipe, la clarté et la lisibilité dans le cas où le produit est relativement simple et que l'équipe est restreinte à deux ou trois développeurs. Choisir MVP ou MVVM est également une bonne stratégie si l'équipe est jeune et peu expérimentée.

Pour des applications plus complexes, avec plus de développeurs et plus de séniorité nous recommandons de **s'inspirer de VIPER (iOS) et Clean Architecture (Android)** qui sont toutes les deux des implémentations de Clean Architecture définies par Robert C. Martin dans *Clean Architecture: A Craftsman's Guide to Software Structure and Design*.

Ces architectures sont plus complexes mais elles permettent de mieux structurer, tester et favorisent le développement par différentes équipes (plusieurs équipes en Feature Teams par exemple).
Les responsabilités sont découpées en cinq parties et chaque partie est indépendante et utilisable par une autre sous forme d'un contrat.

Tests unitaires

L'incontournable bibliothèque Java **JUnit** permet d'écrire des tests unitaires pour Android. L'exécution de ces tests est facilitée par l'intégration du plugin Gradle dans Android Studio, qui permet de les jouer directement depuis l'IDE.

Afin de simuler le comportement de vos composants testés ailleurs dans votre code, vous pouvez utiliser **Mockito**. Cela facilite l'écriture et simplifie vos tests unitaires.

Enfin, étant donnée la complexité du cycle de vie Android, il est parfois nécessaire d'initialiser une Activity ou bien de rechercher des éléments dans une vue. **Robolectric** est alors la librairie à utiliser. Elle simule une partie du fonctionnement du système Android et peut exécuter les tests dans la JVM, permettant de les jouer sur votre machine d'intégration sans rien avoir à ajouter.

Pour écrire vos tests unitaires iOS, vous pouvez utiliser **XCTest**, le framework d'Apple directement intégré dans Xcode. Il permet notamment de réaliser des tests asynchrones ou bien des tests de performance. L'exécution et les rapports de tests sont disponibles en un clic.

Il existe également des bibliothèques tierces. Notons par exemple **Quick**, qui propose une écriture de tests en BDD, rendant beaucoup plus lisibles les actions réalisées par le test. Cette librairie, développée principalement en Swift, mais également compatible avec les projets Objective-C, s'appuie sur **Nimble**, un framework de vérification d'assertions.

Tests fonctionnels

Plusieurs bibliothèques existent pour écrire des tests fonctionnels pour Android.

Tout d'abord, citons **Espresso**, proposée et maintenue par Google. Son principal avantage vient de sa prise en compte du cycle de vie des composants Android. Cela permet d'éviter qu'un test échoue alors que l'application fonctionne bien (*flaky* test). De plus, la bibliothèque est facilement extensible et peut être personnalisée à souhait.

Autre bibliothèque intéressante, **Robotium** qui propose une API, est fortement inspirée de Selenium.

Il existe également des bibliothèques multiplateformes comme **Calabash** ou **Appium**

Pour présenter les résultats des tests fonctionnels, l'outil **Spoon** permet de prendre des captures d'écran au fur et à mesure de l'exécution des tests et les synthétise ensuite sous la forme d'un site Web. Ce site présente chaque test à l'aide d'un slide show d'écrans.

Depuis la sortie de Xcode 7 en 2015, les tests fonctionnels peuvent être conçus directement dans Xcode, grâce à l'outil **UI Tests**. Ce dernier possède de nombreuses fonctionnalités très efficaces, notamment la possibilité d'enregistrer un test depuis le simulateur. Vous n'avez qu'à jouer votre scénario en cliquant sur les interfaces de votre application : l'outil génère le code Objective-C ou Swift du test pour pouvoir le rejouer plus tard. L'exécution de scenarii complexes étant souvent sujette aux erreurs, nous vous recommandons la lecture de documentations tierces[8].

D'autres solutions, comme **KIF**, permettent l'écriture des tests en Objective-C ou en Swift et une compatibilité avec l'outil natif XCTest.

Enfin, la librairie **Appium** se différencie par l'implémentation d'une API compatible avec l'interface de contrôle Selenium WebDriver, un véritable standard du domaine.

8. telles que http://masilotti.com/ui-testing-cheat-sheet/

Monitoring technique et analyse du code

L'outil de référence **SonarQube** pour la qualité de code et la couverture de tests est disponible pour Android. Il est également accompagné d'un outil appelé **Android Lint** spécifique au code Android. L'analyse du style pour Kotlin ne peut pas se faire (pour le moment) via Android Lint, cependant, le langage est bien intégré à l'IDE Celui-ci remonte des informations utiles sur le style du code. Il existe également un coding style pour coder des applications Android en Kotlin[9]. La couverture des tests fonctionnels est calculée par **JaCoCo**.

Jenkins peut automatiser ce monitoring de qualité de code et de couverture de tests unitaires et fonctionnels.

Enfin, outil issu de l'écosystème iOS, l'analyseur statique **Danger** permet de relire automatiquement votre Pull Request ou Merge Request.

Pour la couverture de tests iOS, l'outil **Slather** s'intègre avec **SonarQube**, **Jenkins** ou **Travis CI** en générant des fichiers interprétables par ces outils. Il est compatible avec Objective-C et Swift. **Clang, OCLint, SwiftLint** et **Tailor** permettent d'analyser statiquement les sources des projets. Il existe également des plugins Objective-C et Swift pour SonarQube, avec des versions gratuites (maintenues par la communauté) et payantes (développées par SonarSource).. D'autres outils, comme **Lizard**, pour Swift, permettent aussi le calcul de la complexité cyclomatique de votre base de code.

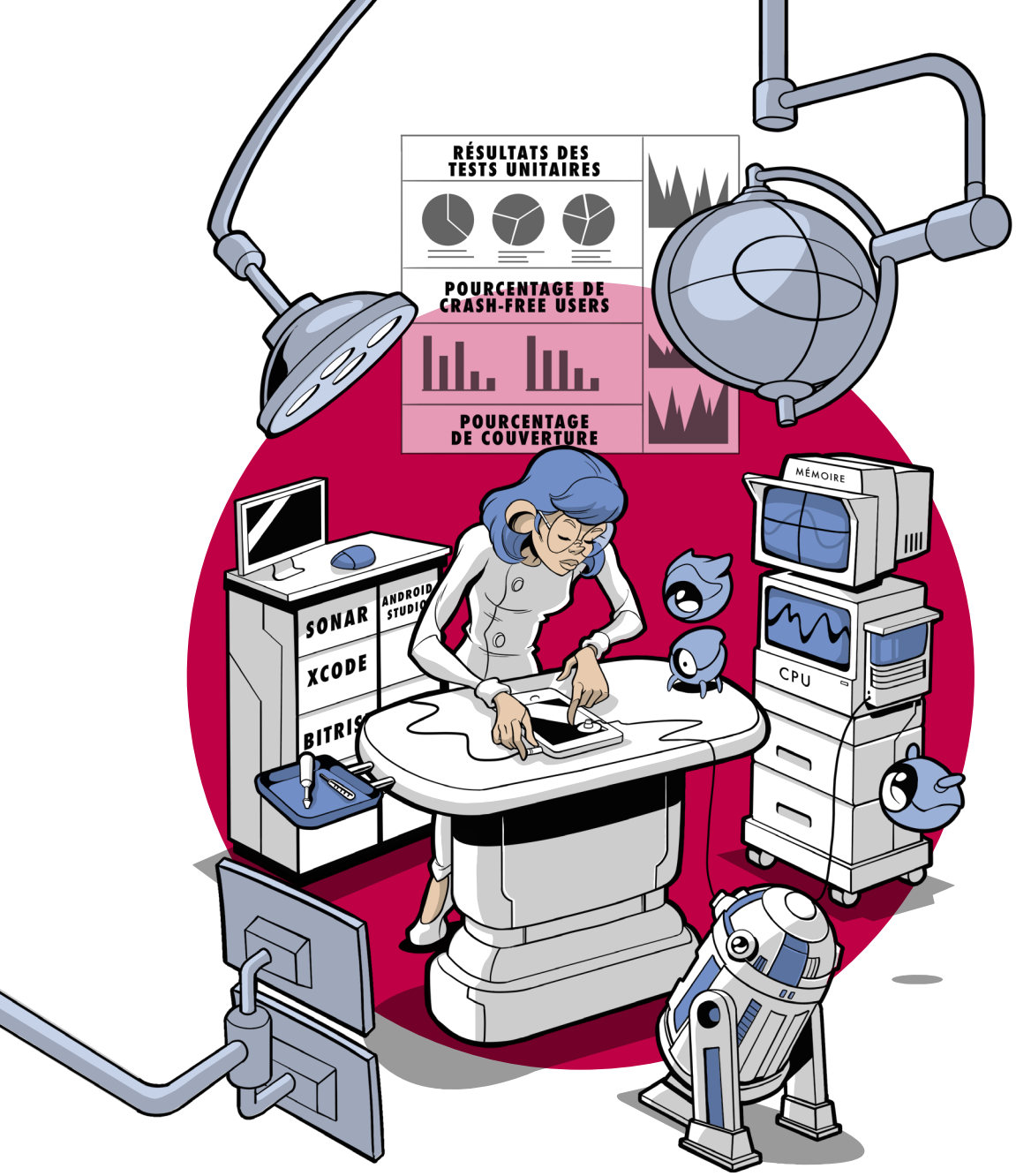
L'intégration de **Danger** rend possible de repérer automatiquement certaines erreurs récurrentes dans votre Pull Request ou Merge Request, telle que la désactivation de tests et d'intégrer le résultat dans une review automatisée sur GitHub, GitLab ou BitBucket.

Monitoring fonctionnel


Le monitoring, élément crucial dans le cycle de vie de votre projet, peut aujourd'hui se faire à l'aide plusieurs outils. **Firebase Crashlytics** propose par exemple des services de suivi de crash fiables et efficaces. Tandis que **Google Analytics**, toujours partie de la suite **Firebase,** met à disposition une console de suivi de tracking et suivi des usages.

Bien qu'aujourd'hui moins usités, les stores officiels, **Play Store** et **iTunes Connect** intègrent également des consoles de suivi basique des métriques de crash et d'adoption de votre application.

9. https://android.github.io/kotlin-guides/style.html (https://bit.ly/2k7rMii)



Monitorer et tester votre application

Intégration Continue : outils de build

Aujourd'hui incontournable dans le cycle de développement d'un projet Android, **Gradle** est l'outil de référence pour la mise en place des tâches de build, testing et signature de votre APK. Grâce à son architecture modulaire, Gradle est facilement extensible et peut donc prendre en charge un nombre illimité de cas d'usage : de la compilation de code Kotlin, jusqu'au déploiement sur Play Store ou Firebase App Distribution.

Moins utilisée, une alternative viable est **Fastlane**, solution officiellement supportée par Firebase.

Fastlane est aujourd'hui la solution de scripting de tâches de build la plus utilisée dans l'écosystème iOS.

Capable de prendre en charge - et de grandement simplifier - signature, compilation et testing des applications iOS, Fastlane se compose également de nombreux outils incontournables, tels que **match** pour gérer certificats et provisioning profile de votre équipe, **snapshot** pour automatiser la capture des écrans d'exemple, **deliver** pour soumettre le binaire à l'App Store, et bien d'autres.

Intégration Continue : environnements


Lorsqu'on souhaite mettre en place un système d'intégration continue, deux choix s'offrent à nous : **SaaS** ou **On Premise**. Dans le premier cas, **Bitrise**, **Circle-CI** et **Travis CI** sont les services les plus utilisés, tandis que **Jenkins**, **GitLab CI** et **Xcode Server** (pour des projets iOS uniquement) sont les plus populaires si vous souhaitez héberger vous même votre outillage.

Q/A de développeurs

Comment consommer facilement une API REST ?	Retrofit	Alamofire, Moya, SwiftHTTP, URLSession
Comment effectuer de l'injection de dépendances ?	Dagger	Typhoon (Objective-C) ou Swinject (iOS)
Comment manipuler efficacement des données JSON ?	Jackson / Gson / Moshi	Moya, Unbox
Quel client HTTP utiliser ?	OKHttp	Alamofire, URLSession
Comment charger et manipuler des images efficacement ?	Glide	AlamofireImage
Comment gérer des tâches asynchrones ?	RxJava 2	PromiseKit ou RxSwift
Comment écrire les logs sur la sortie standard en développement et sur un service externe en production ?	Timber	SwiftyBeaver
Comment stocker de l'information sur l'appareil ?	ORMLite ou Realm	CoreData ou Realm
Comment organiser ses classes en terme d'injection de dépendances ?	Dagger	Swinject
Quel pattern utiliser pour architecturer ses vues et faciliter les tests unitaires ?	MVP et Clean Architecture + Injection de dépendances	MVVM ou VIPER ou Redux + Injection de dépendances
Comment monitorer vos crashs ?	Firebase Crashlytics	
Comment suivre vos utilisateurs pendant vos phases de bêta tests ?	App Distribution de Firebase	
Comment animer son application ?	lottie	lottie, RxAnimated

Take away

Façonner

Appliquer

Appliquer sur les projets mobiles les mêmes recettes de réussite que sur n'importe quel autre projet informatique : agilité, qualité, tests, suivi dans le temps, etc.

Adopter

Adopter les langages modernes que sont Swift et Kotlin dès maintenant sur les nouveaux projets.

Ne pas négliger

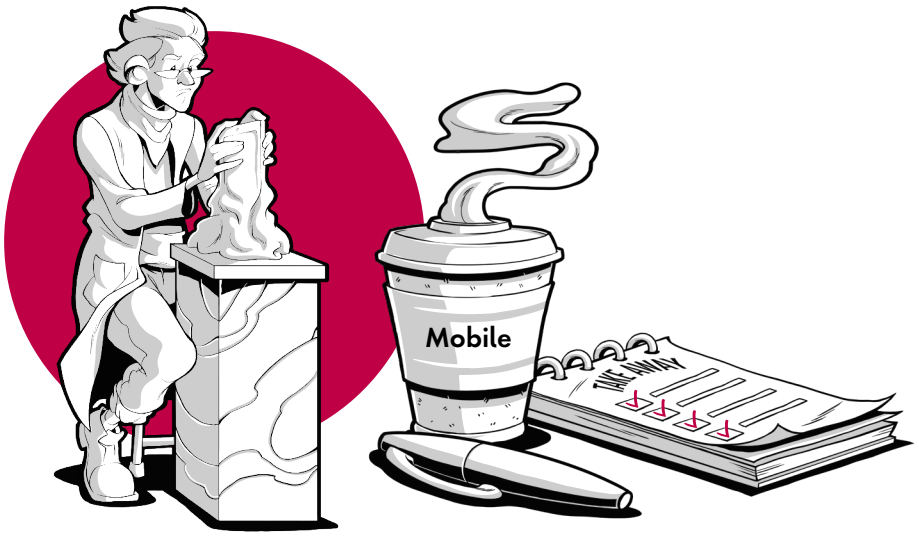
Ne pas négliger les phases d'industrialisation qui présentent des spécificités propres au développement mobile et sont tout aussi importantes, voire plus, que sur un projet « classique ».

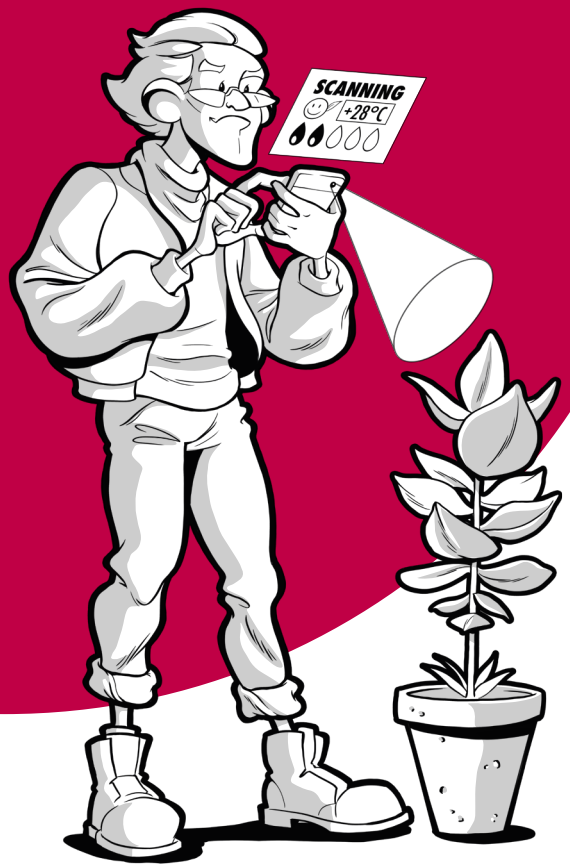
Utiliser

Utiliser les différents canaux de distribution de vos applications tout au long de leur cycle de vie.

Ne pas négliger

Ne pas négliger la culture de la mesure afin de garantir une application techniquement irréprochable et de toujours vous ajuster aux usages.





Innover

Aujourd'hui, la plupart des terminaux mobiles sont plus puissants que les ordinateurs de bureau d'il y a cinq ans.

Cet essor permet de développer des usages de plus en plus autonomes : le terminal possède suffisamment de puissance de calcul pour ne pas dépendre de serveurs distants et donc éviter de coûteux allers/retours sollicitant le réseau.

Sur certains points, le terminal mobile devient même plus performant que le navigateur Web.

Nous explorerons ici trois branches qui ont connu des avancées fracassantes ces derniers mois.

Intelligence Artificielle

L'Intelligence Artificielle (IA) est LA licorne du moment. Tous les géants du Web investissent des millions en recherche sur le sujet. En particulier, le **Deep Learning** (une branche spécifique du Machine Learning et donc de l'IA) a permis de nombreuses avancées déterminantes ces dernières années : reconnaissance d'images, de sons, de texte, génération de nouvelles données, etc. **Les machines talonnent aujourd'hui l'intelligence humaine** dans ces domaines[10].

L'intelligence artificielle est d'ores et déjà à la base de plusieurs fonctionnalités de votre terminal mobile : assistant vocal (peu importe son nom), application galerie photo, application musicale ou encore de fitness.
Bien sûr, vos réseaux sociaux préférés l'utilisent depuis des années pour vous proposer des insertions personnalisées en fonction de vos préférences, de vos habitudes et de votre contexte d'utilisation. La diversité des domaines couverts est extrêmement vaste.

Les assistants vocaux et les chatbots

Apparus pour la première fois en 2011 sur iPhone 4S, les assistants vocaux sont aujourd'hui une fonctionnalité incontournable.

Bien qu'extrêmement limité à l'époque, le domaine a largement bénéficié de la compétition entre les producteurs de systèmes d'exploitation, notamment Apple et Google. Google Assistant et Siri, mais aussi Amazon Alexa et Microsoft Cortana, sont encore loin de proposer une conversation passant le test de Turing mais ils répondent à la plupart des questions que nous leur posons quotidiennement.

Restreints dans un premier temps aux fonctionnalités de base du terminal, les assistants vocaux s'ouvrent aujourd'hui à l'ensemble des applications mobiles : grâce aux **frameworks SiriKit et Google Assistant**, il est désormais possible de configurer une application afin de mettre en place un nouveau point d'interface basé sur l'assistant intelligent. Il est ainsi possible de demander vocalement, en langage naturel, à un assistant d'invoquer des fonctions de notre application, l'envoi d'un message ou la réservation d'une place de restaurant ou d'une course en co-voiturage.

Différence notable entre les univers Apple et Google, Siri est à l'heure actuelle le seul assistant à fonctionner « offline ». L'implémentation des réponses avec SiriKit se fait directement dans le code de l'application mobile là où Google Assistant s'appuie sur **DialogFlow, un outil de reconnaissance vocale externe en SaaS**. Il est donc possible d'implémenter des fonctionnalités d'assistance vocale en mode déconnecté sur iOS, mais pas sur Android. D'autre part, le traitement de la donnée avec SiriKit est gratuit (puisqu'exécuté directement sur le dispositif mobile) alors qu'en fonction de l'utilisation qui en est faite, le service DialogFlow ne le sera pas nécessairement.

Une fois l'action développée, elle sera disponible sur Apple Watch et HomePod dans le cas de Siri et sur l'ensemble des systèmes le supportant pour Google Assistant (Android, Google Wear, Google Home et l'application Google Assistant pour iOS).

10. https://www.publicissapient.fr/services/engineering

Machine Learning

Le Machine Learning est une branche de l'Intelligence Artificielle qui se concentre sur **l'implémentation de méthodes d'apprentissage par l'expérience** permettant à une machine d'exécuter à terme une tâche sans que celle-ci soit explicitement programmée.

Une des applications les plus répandues du Machine Learning concerne la classification, c'est-à-dire l'affectation d'une donnée source à une classe. L'exemple le plus parlant est sans doute la reconnaissance d'images.

Dans le cadre de la mobilité, les opérations de prédiction à partir d'un modèle entrainé étaient encore récemment exclusivement opérées en ligne par les services de la couche serveur de l'application. Grâce notamment à l'évolution de la puissance des terminaux mobiles, il est désormais possible **d'exploiter votre modèle hors ligne** ce qui permet de réaliser vos prédictions directement à bord du dispositif nomade. Il n'est par exemple plus nécessaire d'interroger des serveurs distants pour reconnaître le visage de votre petit neveu sur vos dernières photos de vacances.

Ce changement de paradigme permet d'accélérer drastiquement les prédictions (puisque ces dernières ne sont plus soumises à la latence réseau) ouvrant ainsi la voie à de nouveaux cas d'usage comme l'application d'un modèle au flux vidéo de l'appareil pour y appliquer des filtres ou y effectuer de la reconnaissance d'image en temps réel.

L'autre avancée majeure apportée par l'exécution locale des prédictions concerne **la confidentialité et la protection des données traitées** puisqu'il n'est plus nécessaire de les faire sortir du terminal.

Aujourd'hui, les fonctionnalités liées à l'apprentissage automatique deviennent de plus en plus répandues. Depuis 2017, iOS et Android mettent à disposition des développeurs des solutions logicielles permettant l'exécution hors-ligne de modèles entraînés : il s'agit notamment des **frameworks Core ML d'Apple pour iOS et TensorFlow Lite de Google pour Android**. La plus-value de ces solutions réside dans la simplification de la mise en place des fonctionnalités d'apprentissage automatique (sans connaissance approfondie dans le domaine) tout en exploitant pleinement les ressources du terminal.

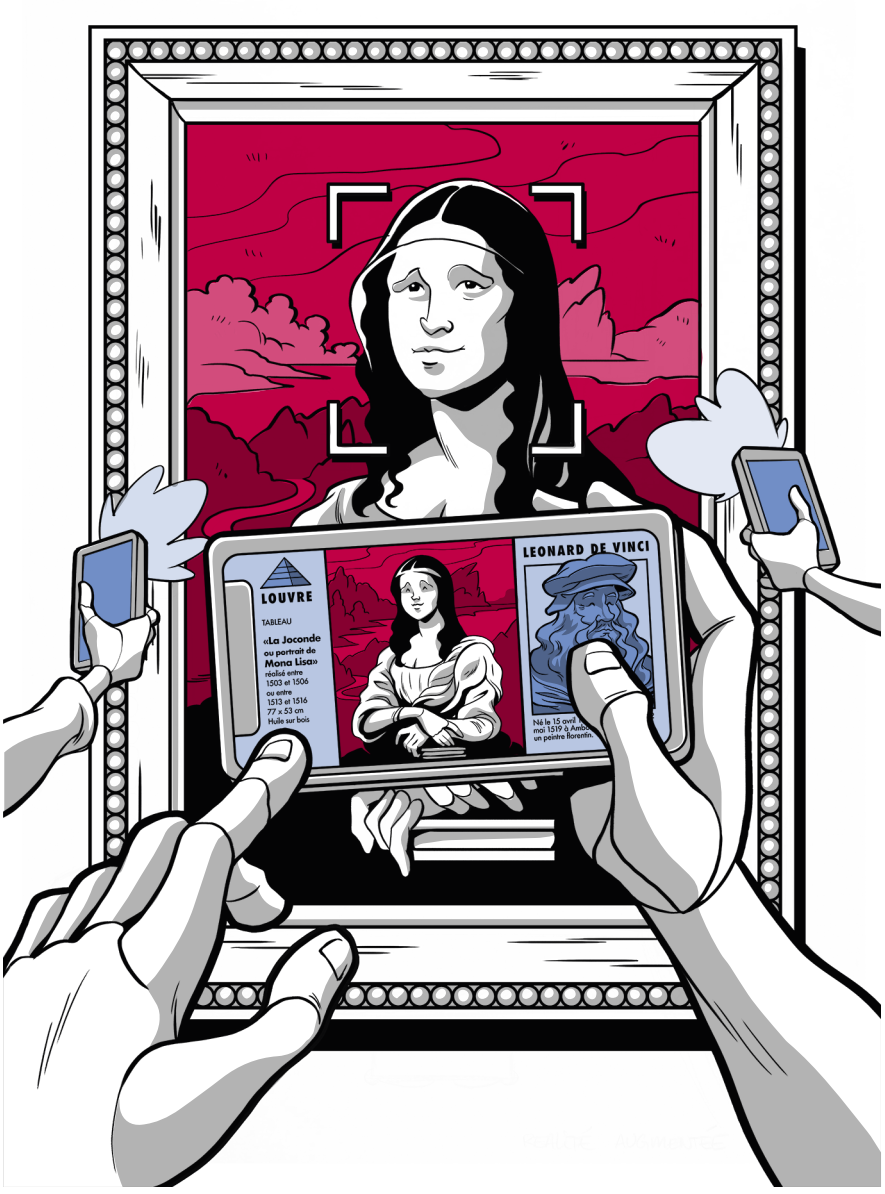
Réalité Augmentée

“ Le Deep Learning a permis de nombreuses avancées déterminantes ces dernières années. ”

Revenue à la mode en 2017, la Réalité Augmentée sur mobile est aujourd'hui une fonctionnalité dont aucun fournisseur de systèmes d'exploitation ne veut se priver : Tango, ARKit, ARCore, à chacun son framework !

La qualité est enfin au rendez-vous et permet d'atteindre la promesse initiale : **transformer votre téléphone en un viseur** qui enrichit ce que vos yeux observent en y ajoutant du contenu de synthèse (bidimensionnel ou tridimensionnel). Un système de capteurs et de logiciels permet de reconstruire le modèle tridimensionnel de l'espace qui entoure l'utilisateur allant jusqu'à reconstruire des informations telles que la direction de la lumière afin de calculer les ombres projetées par les éléments incrustés dans le décor réel.

Les cas d'usages concernent principalement la mise en contexte d'objets fictifs, la prise de mesures sans instrument additionnel, la navigation, les manuels d'assemblage, l'assistance à interventions techniques, l'apprentissage des langues et bien sûr le jeu vidéo.
Ce qui surprend souvent les utilisateurs, ce sont la fluidité du rendu et la qualité visuelle de l'expérience.



Cas d'usage de la réalité augmentée

Solutions techniques

Les deux géants américains de l'IT, Apple et Google, ont commencé à investir considérablement sur la technologie avec **ARKit pour les devices Apple et ARCore pour les devices Android.**

Concernant les couches de rendu (c'est-à-dire le niveau qui prend en charge le dessin des contenus de synthèse), les deux solutions pourront, à leur tour, se baser sur SpriteKit (dessin 2D), SceneKit (dessin 3D) ou Unity (dessin 2D/3D) pour iOS et sur Android NDK ou Unity (dessin 2D/3D) pour Android.

Si d'un côté la réalisation d'images de synthèse de qualité irréprochable demandera des ressources dédiées à la tâche (remettant à l'honneur le travail des graphistes), l'intégration de ces contenus peut être **réalisée directement par les développeurs mobiles** via l'utilisation d'un ou plusieurs frameworks offrant une interface programmable de haut niveau abstrayant les aspects les plus complexes du développement 3D.

En outre, même si la réalisation d'expériences de réalité augmentée nécessite des connaissances spécifiques pour chaque plateforme, il est possible de mutualiser une partie de ces développements (principalement ceux qui concernent le rendu en utilisant Unity).

Terminaux supportés

Actuellement, les terminaux ayant des performances suffisantes pour tirer pleinement parti de ces technologies sont peu nombreux. En effet, ARKit est officiellement supporté à partir des générations d'iPhone sorties en 2017, tandis que la liste fournie par Google contient à ce jour 31 modèles[11].

" La qualité est enfin au rendez-vous et permet d'atteindre la promesse initiale : transformer votre téléphone en un viseur "

11. https://developers.google.com/ar/discover/supported-devices (https://bit.ly/2L3Ge3e)

Le mobile s'immisce partout

Le succès des terminaux mobiles provient en partie de leurs systèmes d'exploitation caractérisés par une faible consommation énergétique et de hautes performances en condition de ressources limitées. Android et iOS se prêtent naturellement à d'autres déclinaisons : téléviseurs, assistants vocaux physiques, domotique, voire vêtements connectés.

Cette polyvalence offre la possibilité aux développeurs d'exploiter un environnement familier, car basé sur le même outillage qui accompagne le développement purement mobile.

Android Things, dédié aux dispositifs IoT, Apple TV et Android TV, les montres connectées et l'interaction avec les systèmes d'une automobile font partie des exemples les plus réussis.

Android Things

Présenté en 2015, Android Things a progressé rapidement en supportant un nombre conséquent de cas d'usages et d'extensions. La promesse de valeur d'Android Things est évidente : **fournir un système s'installant facilement sur les ordinateurs à carte unique** les plus communs, tels que le Raspberry Pi, tout en exploitant facilement la quantité extraordinaire de librairies, outils et services dont l'écosystème Android se compose. Le hardware supporté est pour l'instant limité à six systèmes[12] dont quatre utilisables en production.

Il sera donc simple d'implémenter une solution robuste tirant parti de services et APIs haut niveau proposés par Google pour Android tels que **Firebase, TensorFlow Lite, Google Cloud, Google Assistant, Maps et bien d'autres.**

Pour les développeurs Kotlin, Android Studio, Gradle et la vaste majorité des librairies tierces intègrent naturellement l'écosystème et permettent de se rapprocher du développement Android pour mobile. Il est même envisageable de partager certaines fonctionnalités entre les deux mondes (mobile et IoT).

Néanmoins, Android Things ne pourra pas partir seul à la conquête du marché de l'IoT. En effet, même s'il fonctionne sur des ordinateurs simples, la gestion énergétique des calculs complexes, des senseurs et surtout de la connectivité, reste un enjeu majeur des objets connectés. Il est parfois plus simple de faire tourner du code assembleur sur une puce que de bâtir un système incluant un OS entier. Par conséquent, Android Things se prête majoritairement à une utilisation en tant que gateway locale ou embarquée à l'intérieur d'un matériel d'électroménager (alimenté électriquement de manière stable).

Wearables

Le premier objet connecté notable est indéniablement la montre (ou bracelet) connectée. Elle offre un prolongement des fonctionnalités disponibles sur le téléphone tout en intégrant des capteurs générant des informations de « santé ».

12. https://developer.android.com/things/hardware/ (https://bit.ly/2ma1ywd)

À nouveau, Apple et Google dominent le marché grâce à **Apple Watch et son watchOS et Google Wear OS**. Les fonctionnalités proposées par les deux solutions sont extrêmement similaires : applications, paiement, notifications, suivi de la position GPS et, bien évidemment, assistant vocal.

Malgré le réel succès commercial des *smart watches*[13], les applications natives ne sont pas ancrées dans le quotidien des utilisateurs : à titre d'exemple, les logiciels wearable de Google, Twitter, eBay et Amazon, lancés en 2015, ont été décommissionnés dès 2017.

La taille d'écran limitée **impose une simplicité extrême et des parcours utilisateur simplissimes**. Avant de vous lancer, il faudra donc bien réfléchir à vos usages « montre », sans chercher à y déporter l'ensemble des fonctionnalités de vos applications mobiles. Les cas d'usage les plus porteurs restent aujourd'hui ceux associés aux indicateurs fitness et santé ainsi qu'à la présentation d'informations push enrichies. Les *complications*, voyants affichés sur la montre permettant aux utilisateurs de garder constamment un oeil sur une information importante (le résultat d'un événement sportif, le statut d'une réservation, etc.) peuvent également fournir une excellente déclinaison de votre application.

D'un point de vue de la réalisation logicielle, tout comme pour Android Things, l'outillage dédié aux développeurs est le même que celui utilisé traditionnellement pour les applications mobiles. Google Wear OS et watchOS sont des versions miniaturisées d'iOS et Android. Les **paradigmes restent similaires** même si l'implémentation des vues présente quelques spécificités à ne pas sous-estimer. En ce qui concerne l'installation des logiciels sur la montre, cela passera par le téléphone de l'utilisateur véritable centre de contrôle pour toute configuration de l'accessoire *wearable*.

TV

Au centre de la vie quotidienne de millions d'utilisateurs, le téléviseur a bien sûr été investi par les géants de l'IT.

Avec Android TV, vous pouvez désormais décliner votre application mobile sur TV pour une expérience multi-écrans. Il existe aujourd'hui bon nombre d'appareils exécutant le système d'exploitation *made in Mountain View* : les exemples les plus notables sont ceux des téléviseurs Sony et Philips, des *Set-Top Box* (boitiers) Nvidia Shield ou des opérateurs télécom français Bouygues et Free.

Côté Apple, l'Apple TV, basée sur une variante d'iOS nommée tvOS, est l'une des 4 plateformes faisant l'objet de mises à jour régulières de la part d'Apple. Comme de tradition pour les plateformes made in *Cupertino*, tvOS est uniquement disponible sur Apple TV, vendue par Apple ou distribuée en France dans le cadre des abonnements Canal+.

D'un point de vue des fonctionnalités, Android TV et Apple TV mettent logiquement en avant l'installation d'applications de vidéos et de jeux. Parmi les cas d'usages les plus évolués, Android permet de **pousser des suggestions de contenu sur le fil d'accueil de la TV**, tandis que le boitier **Apple joue le rôle de hub pour les objets connectés** : le téléphone peut donc contrôler les appareils domestiques via l'Apple TV grâce à HomeKit intégré nativement à bord du système.

Là encore, les systèmes TV mettent à disposition le même outillage exploitable pour la réalisation d'applications mobiles : Android Studio, Java ou Kotlin pour Android, Xcode, Objective-C ou Swift pour iOS. De manière disruptive pour la marque à la Pomme, son SDK pour tvOS permet également la création d'interfaces à l'aide de JavaScript et HTML, via TVMLKit.

13. https://www.forbes.com/sites/paullamkin/2018/02/22/smartwatch-popularity-booms-with-fitness-trackers-on-the-slide (https://bit.ly/2lF1Qed)

Android Auto / CarPlay

Le dernier secteur en date à profiter de l'omniprésence des mobiles est l'automobile. Il est désormais possible d'obtenir de nombreuses informations sur l'état et les statistiques du véhicule directement depuis votre mobile. Vous pouvez contrôler l'allumage du moteur ou le chauffage à distance ce qui peut s'avérer très pratique en hiver - voire terriblement dangereux comme l'ont prouvé quelques hackers qui ont pu prendre le contrôle d'une Jeep sur l'autoroute[14]. La localisation du véhicule ou encore son prêt peuvent maintenant se gérer sur un smartphone grâce à des services tels que Virtuo ou OuiCar.

Les deux géants s'y intéressent d'ailleurs de près et ont créé **Android Auto et CarPlay**. Intégrés nativement au tableau de bord du véhicule (et faisant donc l'objet d'une féroce guerre commerciale auprès des constructeurs), ils permettent le contrôle de la musique, la navigation et, bien sûr, des assistants vocaux. De plus en plus de constructeurs de véhicules proposent aujourd'hui des **écrans généreux destinés à héberger des applications de *In-Car Infotainment***.

Néanmoins, leur développement n'a rien à voir avec la réalisation d'applications wearable ou TV : la logique des logiciels Android Auto et CarPlay est **entièrement pilotée par le smartphone** (l'ordinateur de bord du véhicule ne doit opérer aucun traitement), qui exploite le display de l'automobile de façon similaire à celui d'un écran externe. Les interfaces présentées seront donc dictées par le smartphone.

> " Il est simple d'implémenter une solution robuste tirant parti de services et APIs haut niveau proposés par Google pour Android. "

14. http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ (https://bit.ly/1MFdhVm)

Take away

Innover

Profiter

Profiter des capacités de traitement des nouvelles générations de mobile pour protéger, *by design*, les données privées de vos utilisateurs.

Projeter

Projeter les utilisateurs dans un monde « augmenté » en adoptant dès maintenant les usages AR.

Déployer

Déployer les applications sur une grande variété de supports afin d'accompagner les utilisateurs tout au long de la journée, où qu'ils soient.

Conclusion

Le smartphone est devenu l'accessoire central de la vie d'un grand nombre de nos concitoyens. Aujourd'hui, produire une application distribuée sur les principaux stores est à la portée de n'importe quelle entreprise, de la PME à la multinationale. Faire perdurer cette application dans le temps et l'installer durablement dans le paysage est un tout autre enjeu. Grâce à l'offre pléthorique dont ils disposent, les utilisateurs sont de plus en plus versatiles et de moins en moins enclins à pardonner une faute.

L'expérience que vous leur offrez se doit donc d'être irréprochable. Cela passe par des mises à jour fréquentes pour rester au goût du jour mais aussi nécessairement par une application pensée pour l'usage mobile et non pas par une simple transposition de vos usages Web.
Dans cette quête de la perfection, vous devrez arbitrer : **offrir une expérience native** est ce qui se fait de mieux dans chacun des mondes Android ou iPhone. Cependant, cette perfection a un coût qui n'est pas nécessairement pertinent dans toutes les situations.

L'autre changement majeur dans le monde du mobile est qu'il ne suffit plus d'être le premier sur le marché pour s'installer. Pour durer, votre logiciel **doit être traité comme un véritable asset** : vous devez le construire pas à pas pour suivre les tendances du marché, vous devez être capable de le livrer fréquemment et sereinement et de pouvoir attirer les meilleurs talents de l'écosystème en leur offrant des environnements de conception et de développements modernes. Dans le mobile, comme ailleurs, le **déploiement continu (et la rigueur logicielle qu'il implique) est aujourd'hui une réalité** dont vous ne pourrez pas vous passer si vous voulez vous tailler la part du lion.

Enfin, nos smartphones ne sont pas qu'un écran de plus à notre disposition. Ils participent aujourd'hui à tous les aspects de notre vie quotidienne : analyse de nos données de santé, nouvelle paire d'yeux ou d'oreilles, voire second cerveau. La puissance actuelle des terminaux de dernière génération leur permet de devenir ce **véritable assistant personnel** qu'on nous promettait lors de la sortie des premiers Palm. La **réalité augmentée** modifie notre vision du monde, les **assistants vocaux** se tiennent prêt à exécuter le moindre de nos désirs et les **applications de Deep Learning** nous offrent des aides à la décision parfois plus malignes que nos propres intuitions. Les technologies mobiles s'immisçant de nos téléviseurs à nos voitures, leur omniprésence ne devrait que s'accentuer.

Vous l'aurez compris, être présent dans un store aujourd'hui ne suffit plus : il faut être **désirable, irréprochable techniquement et innovant** que ce soit dans les usages tout comme dans la technique. Le terrain de jeu pour les développeurs mobiles d'élite est de plus en plus vaste !

Take away

Mobile

Comprendre

- Réaliser son application spécialement pour le mobile que ce soit au niveau des fonctionnalités ou de l'UX. Ne pas réaliser un simple portage.
- Choisir les technologies de développement en analysant finement tous les paramètres pas seulement le coût.
- Adapter son SI afin de servir au mieux les usages « nomades ».

Façonner

- Appliquer sur les projets mobiles les mêmes recettes de réussite que sur n'importe quel autre projet informatique : agilité, qualité, tests, suivi dans le temps, etc.
- Adopter les langages modernes que sont Swift et Kotlin dès maintenant sur les nouveaux projets.
- Ne pas négliger les phases d'industrialisation qui présentent des spécificités propres au développement mobile et sont tout aussi importantes, voire plus, que sur un projet « classique ».
- Utiliser les différents canaux de distribution de vos applications tout au long de leur cycle de vie.
- Ne pas négliger la culture de la mesure afin de garantir une application techniquement irréprochable et de toujours vous ajuster aux usages.

Innover

- Profiter des capacités de traitement des nouvelles générations de mobile pour protéger, by design, les données privées de vos utilisateurs.
- Projeter les utilisateurs dans un monde « augmenté » en adoptant dès maintenant les usages AR.
- Déployer les applications sur une grande variété de supports afin d'accompagner les utilisateurs tout au long de la journée, où qu'ils soient.

À lire et à relire

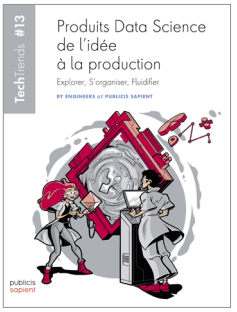
Pour télécharger l'un des TechTrends en version éléctronique (pdf), rendez-vous sur le site publicissapient.fr/services/engineering
Pour demander une version papier, envoyez un mail à techtrends@publicissapient.com

Pour en savoir plus sur le Data, le Cloud, le Web, les architectures Java, la mobilité et l'agilité, rendez-vous sur blog.engineering.publicissapient.fr

Techtrends #9
Cloud
Préparer sa migration, Séléctionner son offre, Choisir ses technologies

Techtrends #10
Studio
Démarrer son projet, Piloter son projet, Pérenniser son produit

Techtrends #12
Conteneurs
Projeter, Implémenter, Exploiter



Techtrends #13
Produits Data Science
Explorer, S'organiser, Fluidifier

Techtrends

#11 Internet of Things
#7 Back
#6 DataLab
#5 Front
#4 Craftsmanship
#3 Agilité
#2 DevOps
#1 Data

Autres parutions

Scrum Master Academy
Le Guide du Scrum Master d'élite

Les Communautés de Pratique en Pratique
Le Mini Guide

Product Academy
Le guide des Product Managers et des Product Owners d'élite

Les auteurs

Simone CIVETTA

Nicolas THENOZ

Pablo LOPEZ

Benjamin LACROIX

Qian JIN

Florent CAPON

Michaël OHAYON

Christophe HEUBÈS

Direction artistique : **Publicis Sapient Engineering**
Design graphique : **micheletmichel.com** / Illustrations: **lasaintepaire.com**

Publicis Sapient
94 Avenue Gambetta, 75020 Paris
+33 (0)1 53 89 99 99
engineering@publicisapient.com

Toutes les informations sur :
publicissapient.fr/services/engineering