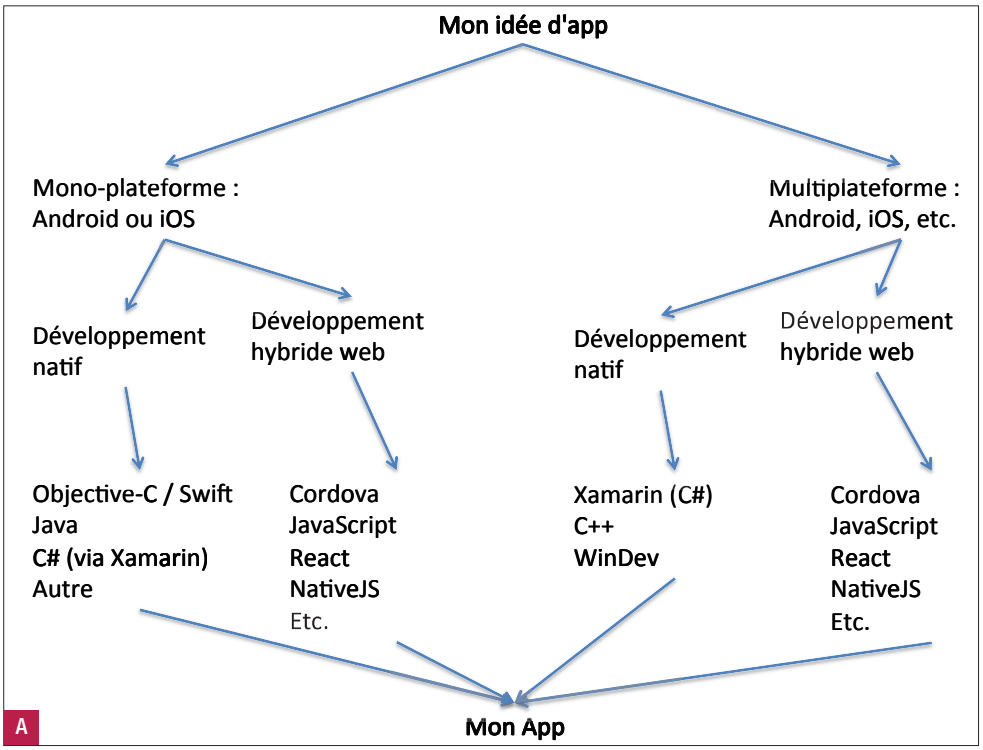
Développement mobile : choisir, et si possible bien choisir

En 10 ans, le développement mobile a totalement changé que ce soient les matériels, les systèmes et les outils de développement. Au début, nous étions en natif, natif et encore natif. Puis nous sommes passés à l'hésitation multiplateforme avec des outils plus ou moins fiables, et aux apps Web. Aujourd'hui, où en est-on ?



A

Le développeur, qu'il soit débutant ou expert, n'a que l'embarras du choix, et c'est bien là le problème. Quel outil choisir ? Schématiquement, nous pouvons résumer le premier niveau de réflexion à un diagramme très simple : [A]

En y réfléchissant un peu, nous allons vite nous apercevoir que de nombreuses questions se posent derrière et qu'elles sont tout aussi fondamentales :

- Est-ce un développement "loisir" ou un projet de grande envergure ?
- Est-ce une app déployée sur les Apps Stores pour tout le monde ou une app plutôt interne à l'entreprise ?
- Quelles sont mes compétences ?
- Quel budget ai-je à ma disposition ? Faut-il investir dans des formations, des programmes développeurs ou dans les outils ?
- De quelles fonctions ai-je besoin dans l'app ? Est-ce que je dois beaucoup utiliser le matériel ou non ?
- L'interface est-elle très spécifique ou très généraliste ?
- Mon app est-elle monolangue ou multilangue ?
- Est-ce que je travaille seul ou en équipe ? Dans ce cas, quelle gestion de projets faut-il mettre en place ?
- Quels sont les tests terminaux à faire ? Et avec quels outils ?
- L'app aura-t-elle un cycle de vie régulier (mise à jour, maintenance, etc.) ?
- Me faut-il un backend et des services backend spécifiques (bases de données, stockage, génération 3D, authentification, etc.) ?

Ça va ? Pas trop mal à la tête ? Nous n'aborderons pas dans ce dossier tous les services et fonctions backend qui nécessitent souvent d'intégrer des services externes (typiquement en mode Cloud) comme pour l'authentification (ex. : authentification Facebook), stockage via un stockage Web, un traitement d'analyse en mode Cloud, calcul et affichage d'une interface dynamique (ex. : jeux mobiles).

Après il faut raison garder. On ne va pas coder une "simple" app de gestion d'une entreprise avec les fonctions les plus avancées d'interface (ex. : WebGL, Unity3D) ou coder en C++ une simple app avec une base de données, ni utiliser SQL Server ou Oracle Database quand MySQL ou SQL Lite suffit largement. Bref, n'utilisez pas un tank quand un vélo suffira.

Des coûts à considérer

Tout développement a un coût, un budget. Par exemple, si une société Y fait développer une app mobile, elle doit considérer le salaire du développeur (qu'il soit interne ou externe). Et selon le profil du développeur, et des outils utilisés, la facture ne sera pas la même. Après, attention aussi : un profil performant a un coût généralement supérieur à la moyenne mais vous pouvez gagner en temps de développement…

Cet article détaille le coût du développement même si l'étude est centrée sur les USA ; cela peut donner une bonne idée : http://www.comentum.com/mobile-app-development-cost.html

Sur les salaires des développeurs mobiles, les profils Android et iOS ont toujours une majoration. Ainsi un profil débutant peut espérer +35 k€, voire, 40 k€ / an. Mais nous trouvons certaines fourchettes salariales trop extrêmes : 42-45 k€ en profil 0-2 ans expérience. La tendance actuelle est clairement d'avoir un profil multiplateforme et non plus sur une seule plateforme. •

Développement mobile : quelles solutions, comment choisir, quel outil pour quel développement ?

Arnaud Piroelle
@ArnaudPiroelle
Consultant Android,
Xebia

Florent Capon
Consultant iOS,
Xebia

Maxime Fontanier
Maxime.Fontanier@cnes.fr
Service DNO/DA/AQ
CNES

Sylvain Teodomante
Sylvain.Teodomante@cnes.fr
Service DNO/DA/AQ
CNES

Alors qu'il y a quelques années, nous pouvions encore nous poser la question, aujourd'hui c'est bien une réalité : le monde est mobile. Une donnée le prouve : en 2014, les ventes de terminaux mobiles ont dépassé celles des PC. Les utilisateurs naviguent davantage sur leurs smartphones et tablettes, et deviennent de plus en plus exigeants sur les performances, le temps de réactivité, l'ergonomie de l'interface, etc. Il devient indispensable d'intégrer cette évolution et de développer des applications de qualité.

Pour ce faire, il existe deux approches :

- Le web mobile : il s'agit de développer un site Web adapté au mobile, plus connu sous le nom de « responsive ». On parle également de « progressive WebApp » permettant d'optimiser le chargement des écrans mais aussi la gestion du mode offline.
- L'application mobile : l'application, spécifique au mobile, n'a pour cible que le smartphone, la tablette et la montre connectée. Son avantage est de proposer une interaction forte avec toutes les possibilités offertes par le hardware (appareil photo, gyroscope, etc.) et d'être présent dans les stores de chaque plateforme choisie (au contraire d'un site Web mobile).

Dans le cadre d'une approche Web Mobile, le choix de la technologie est large puisque identique au développement Web « traditionnel ». Il existe aujourd'hui de nombreux frameworks HTML / CSS / JS (dont notamment Angular, React.js ou Vue.js) pour développer un site mobile ; la plupart proposant des outils pour faciliter l'affichage responsive et les interactions avec le hardware. Ces réalisations ne représentant pas un « développement mobile » en tant que tel nous ne l'aborderons pas davantage au cours de cet article.

Dans la seconde approche, le choix de la technologie peut être plus délicat car les solutions possibles (native, hybride, transcompilée) présentent des orientations de développements très différentes avec des avantages et inconvénients marqués. Ce choix crucial impactera par la suite la performance, la maintenabilité, la pérennité, le coût, etc.

Développement

Trois solutions : native, hybride, transcompilée

Pour réaliser une application mobile qui a pour objectif d'être déployée sur un store (qu'il soit public ou d'entreprise), trois solutions techniques sont possibles :

- Les technologies natives : elles permettent de développer une application spécifique pour chaque plateforme (iOS, Android, Windows Phone), dans un langage supporté officiellement par l'éditeur de la plateforme.
- Les technologies hybrides : elles permettent d'encapsuler un site Web dans une application « coquille » native, afin de faciliter l'accès aux composants du dispositif à partir du code Web.
- Les technologies transcompilées : elles permettent de développer une seule application à l'aide d'un langage unique qui sera transformée en applications natives, pour chaque plateforme cible.

Quelle solution pour quel besoin ?

Choisir une solution technique pour le développement d'une application mobile n'est pas simple. De nombreux paramètres entrent en jeu, que ce soit le mode de distribution de votre application, les plateformes ciblées ou même les ressources disponibles pour le développement.

Dans le cas d'un déploiement multiplateforme, il n'existe pas de solution miracle, car chacune des technologies a ses forces et faiblesses. Le choix est plutôt une question d'équilibre, entre le temps consacré aux développements, les coûts engendrés, la pérennité de l'application, sa maintenance et la qualité souhaitée.

Pour des réalisations plus petites ou dites « jetables », l'hybride semble être une bonne option. Dans le cas inverse, pour une application plus complexe ayant pour vocation de durer dans le temps, le natif s'impose naturellement.

Pourquoi choisir le natif ?

Le développement natif consiste à développer une application pour chaque plateforme cible. Il est ainsi supporté officiellement par l'éditeur de la plateforme cible : Apple pour iOS, Google pour Android et Microsoft pour Windows Phone. Il est alors aisé de suivre les évolutions de chaque constructeur, OS et API proposés.

Un second avantage et non des moindres, ce sont les communautés de développeurs actives sur ces plateformes (surtout pour iOS et Android). Il est alors facile de trouver des réponses aux problèmes que l'on peut rencontrer. On trouvera aussi des milliers de librairies tierces pour tout type de besoin.

Le développement natif permet aussi de proposer la meilleure expérience utilisateur possible avec des performances optimales : il assure une pleine exploitation de toutes les capacités et ressources proposées par le

device, sans aucune couche intermédiaire venant réduire la mémoire ou les temps de calcul. Dans le cas d'applications à forte sollicitation du CPU ou GPU, la différence entre une application native ou hybride peut être très marquée : scrolling saccadé, freeze ou même crash si la mémoire n'est pas gérée correctement.

Autre grande force du natif : la gestion de l'affichage de listes de plusieurs centaines ou milliers d'éléments ; grâce à des composants dédiés avec un système de recyclage des cellules limitant la sollicitation CPU et mémoire, ce type d'affichage, devenu presque incontournable aujourd'hui, est facile à intégrer.

Nous parlions précédemment d'expérience utilisateur. La performance et la réactivité sont certes des points importants dans ce domaine, mais une autre composante est essentielle : l'ergonomie. En effet, chaque système d'exploitation et device dispose des codes ergonomiques qui lui sont propres. Ainsi, le respect de ces codes décrits par Apple, Google ou Microsoft au travers de documentations, est facilité par l'ensemble des outils natifs mis à la disposition du développeur. Leurs évolutions créent très peu d'impact en termes de temps de développement car sont anticipés par le constructeur.

Enfin, dernier point non négligeable, notamment dans la réalisation d'applications d'envergure visant à être pérennes : la maintenabilité, accrue grâce à une base de code saine écrite et développée spécifiquement pour une plateforme. Le code n'est alors pas pollué par des instructions qui pourraient être différentes d'une plateforme à une autre, contrairement à un développement hybride. Les risques de régression sont ainsi réduits puisqu'un correctif spécifique à une technologie n'aura aucun impact sur les autres plateformes. Les phases de recette et test s'en trouvent aussi allégées. [1]

Pourquoi choisir l'hybride ?

Un développement hybride implique une base de code unique qui sera la même d'une plateforme à une autre. Avantage considérable par rapport à un développement natif, cette base de code unique permettra un développement multiplateforme très rapide. Ceci permet de garantir l'ensemble des fonctionnalités et la maintenance de manière indépendante de la plateforme.

De plus, le développement hybride bénéficie d'un bon nombre de solutions techniques. Il y en a pour tous les goûts. Les technologies venant du monde du Web sont les plus utilisées. Depuis des dizaines d'années, les développeurs du monde entier utilisent HTML, CSS et JS afin d'offrir des sites Web de qualité aux utilisateurs. Rien d'étonnant à ce que ces technologies soient utilisées dans des solutions hybrides sur mobile. Représentant une très grande partie des développeurs sur le marché, il est plutôt aisé de trouver des profils qualifiés sur ces technologies pour réaliser un projet hybride.

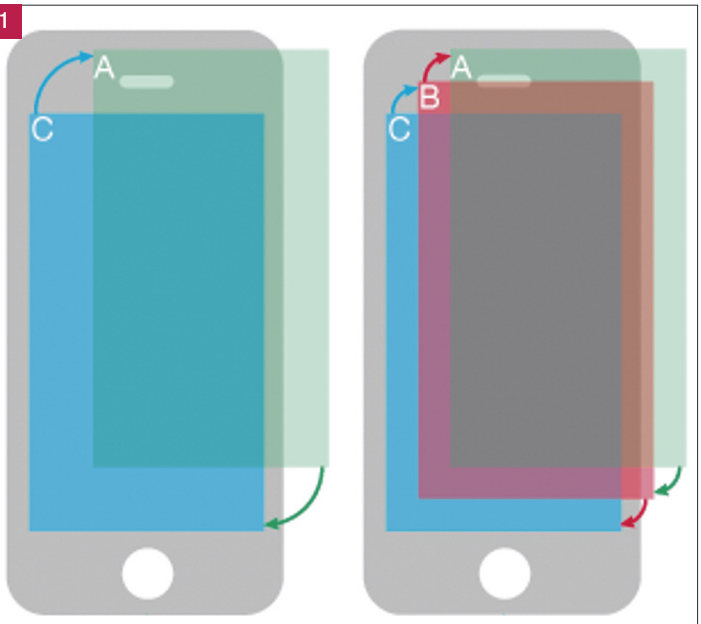
Cependant, ce type de développement a des limites. Que ce soit en termes de performances, de fonctionnalités liées aux hardwares ou de respect de l'expérience utilisateur, une solution hybride ne peut arriver au niveau d'un développement natif.

Pourquoi choisir le transcompilé ?

Au vu des avantages et inconvénients des 2 modes de développement proposés précédemment, de nouvelles solutions visent à garder le meilleur des deux mondes, à savoir une base de code unique avec les performances et les fonctionnalités des différentes plateformes. Parmi elles, on retrouve par exemple Xamarin ou encore AppCelerator.

Ces solutions permettent le développement d'une base de code unique qui sera transformée par la suite en code natif. Elles proposent un support au plus proche des plateformes, facilitant ainsi l'optimisation des performances et le respect de l'expérience utilisateur. Toutefois, ce support n'est pas magique, il faut garder en tête que chaque mise à jour de la plateforme implique une mise à jour du framework. Il faut donc un minimum de temps afin de bénéficier des nouveautés de chaque plateforme.

1



À gauche : développement natif ; à droite : développement hybride
A : La couche OS
B : La couche hybride (webview)
C : La couche physique (l'écran tactile)

Et des webviews dans du natif ?

Impossible de ne pas avoir entendu parler de « webview dans du natif ». Oui c'est possible, oui, ça se fait souvent. Néanmoins, il faut veiller à utiliser cette solution avec parcimonie car elle représente dans bien des cas une « roue de secours » ou un palliatif à un développement trop coûteux.

Même si cela peut présenter des avantages (délai, coût, etc.), les utilisateurs de votre application ne sont pas dupes et constateront un décalage ergonomique et souvent graphique entre vos parties développées en natif et les écrans affichés dans une webview. Si votre projet ressemble à un patchwork de technologies, vos utilisateurs risquent d'être déçus : la notation de votre application dans le store pourrait en pâtir.

Quel langage utiliser ?

Même s'il ne sera pas nécessairement un critère de choix, il peut-être important de prendre en compte quel langage est possible pour chaque solution technique.

Dans le cadre d'un développement natif, plusieurs solutions sont possibles selon les plateformes :

- iOS (Apple) : Swift remplace désormais le vieillissant Objective-C, même si ce dernier est toujours possible mais non recommandé. À noter que contrairement à l'Objective-C, Swift est open source et permet de développer des applications serveurs.
- Android (Google) : Java est le langage officiellement supporté par

Google mais d'autres sont possibles, comme Groovy ou même Scala… Néanmoins, ils ne sont que très anecdotiques car lourds à mettre en place et peu adaptés au développement mobile. Mais un nouveau langage a fait son apparition et peut changer la donne : Kotlin. Conçu par JetBrains en 2014, il est pensé comme un langage fonctionnel permettant de simplifier au maximum le code et de rendre la maintenance plus facile. Bien qu'assez jeune, Kotlin est parfaitement adapté au développement mobile sur Android.

- Windows Phone (Microsoft) : nativement, Microsoft propose trois langages différents. Chacun sera libre de choisir selon ses préférences :
  - C# / XAML : langage très proche de Java, le C# est très reconnu dans le monde du développement embarqué.
  - JS / HTML : JS et HTML sont très populaires dans le monde du Web. L'intégration de ces langages dans le développement mobile est un moyen d'attirer une nouvelle population de développeurs.
  - C++ / XAML : langage de programmation très populaire pour ses performances et son approche très bas niveau.

Dans le cas d'une solution hybride, le développeur pourra choisir le framework HTML / JS / CSS de son choix.

Pour les technologies transcompilées, le langage dépendra de la solution choisie. Par exemple, une application AppCelerator s'écrit en JavaScript alors qu'une application Xamarin se développe en C#.

Quelle stack technique choisir ?

Le monde de la mobilité évolue très vite et il n'est pas rare de voir une stack technique devenir obsolète en quelques mois seulement. Au risque donc d'être dépassés dans les prochains mois, nous avons souhaité vous présenter les stacks techniques recommandées à l'heure actuelle :

	iOS	Android	Windows
Client HTTP	URLSession ou Alamofire	OkHttp	Restsharp
Client API REST	URLSession ou Alamofire	Retrofit	Restsharp
Sérialisation / Désérialisation JSON	Unbox	Gson ou Moshi	Newtonsoft json
Gestion des images	AlamofireImage	Glide	Imaging SDK
Programmation réactive	RxSwift	RxJava	
Gestion des logs	CocoaLumberjack	Timber	Log4Net
Base de données locale	Realm	SQLBrite	SQLite
Injection de dépendances	Swinject	Dagger	MvvmLight
Crash reporting	Crashlytics	Crashlytics	HockeyApp
Tracking utilisateur	Fabric	Fabric	Application Insights

Ces stacks techniques doivent être maintenues à jour et modifiées régulièrement afin de suivre l'évolution de chaque plateforme.

Pour un développement hybride, c'est différent. Nous parlerons davantage de frameworks regroupant tous les outils nécessaires à la bonne réalisation d'une WebApp. Chacun ayant ses spécificités, patterns et architectures, le développeur pourra choisir celui avec lequel il se sent le plus à l'aise. Les plus connus aujourd'hui sont :

- AngularJS
- EmberJS
- React
- Vue.js

Quelle solution pour quel coût ?

Nous venons d'analyser, pour chaque technologie, les points forts et critères importants qui doivent aider à choisir une technologie plutôt qu'une autre. Néanmoins, la question du budget consacré au développement d'application peut contredire ce choix. Là où le natif paraît être la solution garantissant les meilleures performances et la maintenance la plus fiable, elle présente en revanche des coûts souvent plus élevés que ses concurrents. En effet, dans le cas d'une réalisation multiplateformes, le natif oblige l'équipe à développer une version de l'application pour chaque technologie supportée. Le coût de réalisation est alors multiplié par le nombre de plateformes souhaitées. Chaque technologie ayant son propre langage et ses codes ergonomiques, il sera difficile de capitaliser d'une plateforme à une autre. De même, les profils intégrés dans les équipes seront spécialisés sur une plateforme mobile alors que du côté hybride, il est possible pour un développeur full stack de travailler sur des projets desktop et mobiles.

Outre ceci, il faut aussi prendre en considération le coût des IDE, réel avantage budgétaire pour un développement natif. En effet, chaque constructeur propose gratuitement son propre IDE, alors que certaines solutions hybrides ou transcompilées sont payantes pour un usage commercial comme Xamarin ou AppCelerator.

Le coût des licences pour déployer son application sur les stores est quant à lui identique pour chaque type de développement. Pour distribuer une application dans un store (public ou privé), il est nécessaire de souscrire à une licence de développement qu'il s'agisse d'un développement natif, hybride ou transcompilé. À noter qu'Apple présente les tarifs les plus élevés : 99€ / an pour l'App Store à 299€ / an pour un store d'entreprise, contre 25$USD à vie pour Google (soit environ 24€) et de 14€ à 75€ à vie pour Microsoft.

CONCLUSION

De nombreux critères entrent en jeu pour choisir la solution la plus adaptée à son besoin. Or, il n'existe pas de solution miracle. Tout est question d'équilibre entre les différents critères pour pondérer votre choix. Le plus simple reste de synthétiser tous ces éléments dans un tableau :

	Natif *iOS, Android, Windows Phone*	**Hybride** *Apache Cordova, PhoneGap, Ionic, etc.*	**Transcompilé** *Xamarin, AppCelerator, etc.*
Communauté	⊕	⊕	⊖
Compétences sur le marché	⊖	⊕	⊖
Performances	⊕	⊖	⊕
Expérience utilisateur	⊕	⊖	⊖
Capitalisation du code pour d'autres supports	⊖	⊕	⊖
Maintenabilité	⊕	⊖	⊖
Délai de réalisation pour un gros projet	⊕	⊖	⊖
Délai de réalisation pour un petit projet	⊖	⊕	⊕
Coût de réalisation	⊖	⊕	⊕