



Panorama des technologies pour développer une app mobile

GUIDE PRATIQUE

MISE À JOUR 2022



Ceci est un ebook, ça serait chouette
de ne pas l'imprimer ! 

Sommaire

INTRODUCTION	4
Le début d’une nouvelle ère	5
Des tendances qui se confirment	5
Applications natives, compilées, hybrides et web	7

1

LES APPLICATIONS NATIVES POUR UNE EXPÉRIENCE SANS COUTURE	9
---	---

2

KOTLIN MULTIPLATFORM OU FLUTTER : LA REVANCHE DES APPLICATIONS CROSS-PLATFORMES	15
Le partage de code Android vers iOS avec Kotlin Multiplatform Mobile : la mutualisation sans effort ?	17
Flutter : le pari gagné de Google ?	22

3

LE JAVASCRIPT À LA CONQUÊTE DU MOBILE 26

React Native : le « nouveau » de l’hybride 28

NativeScript, le parent pauvre de la convergence 32

4

LE WEB ENCAPSULÉ 33 DANS LES APPLICATIONS MOBILES

Les webviews, toujours d’actualité, toujours utiles 34

Les PWA et TWA : une nouvelle incursion du web 35
dans nos smartphones

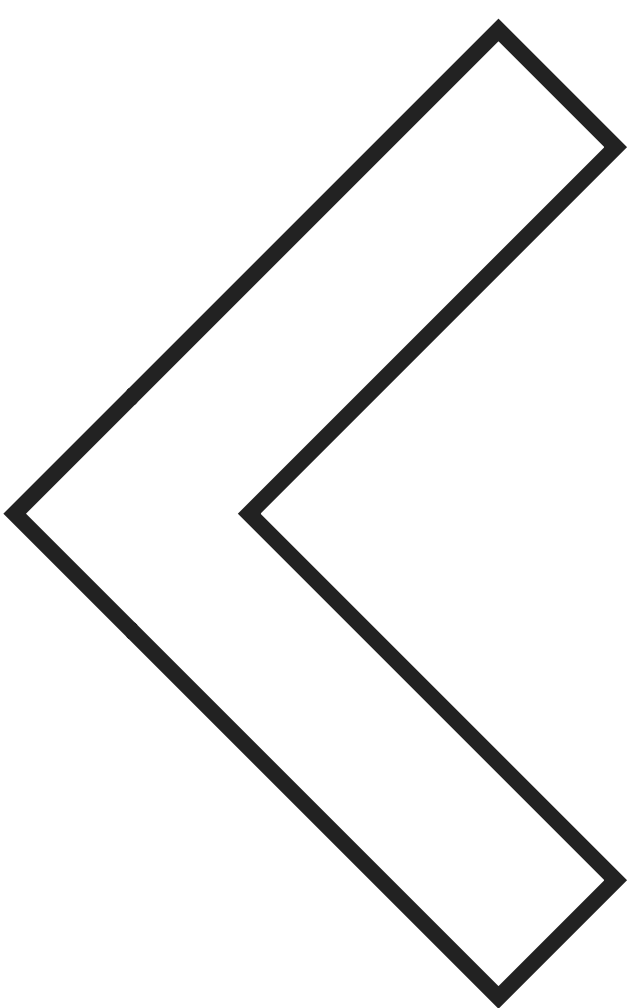
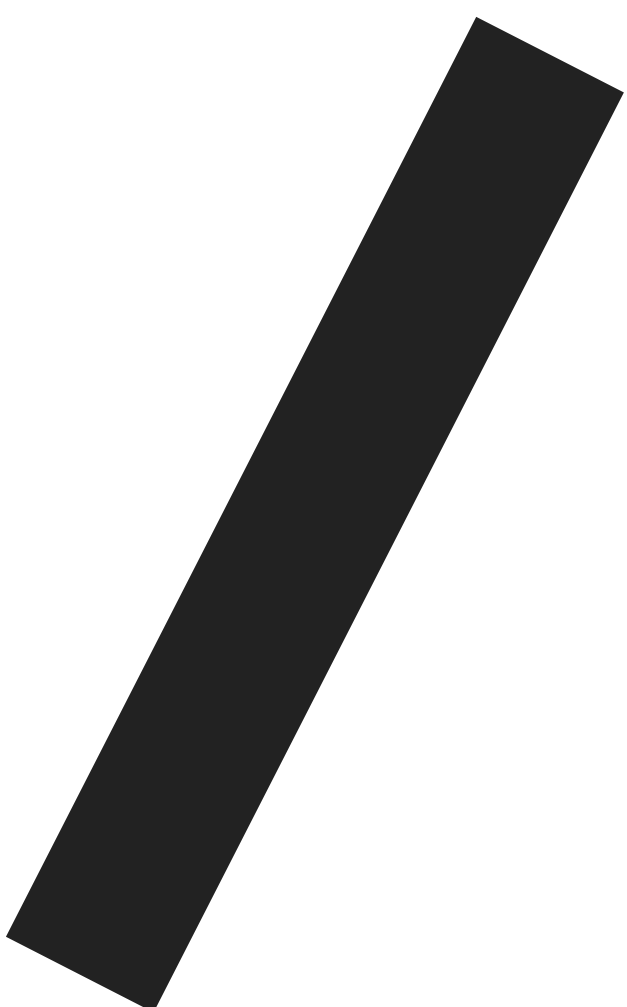
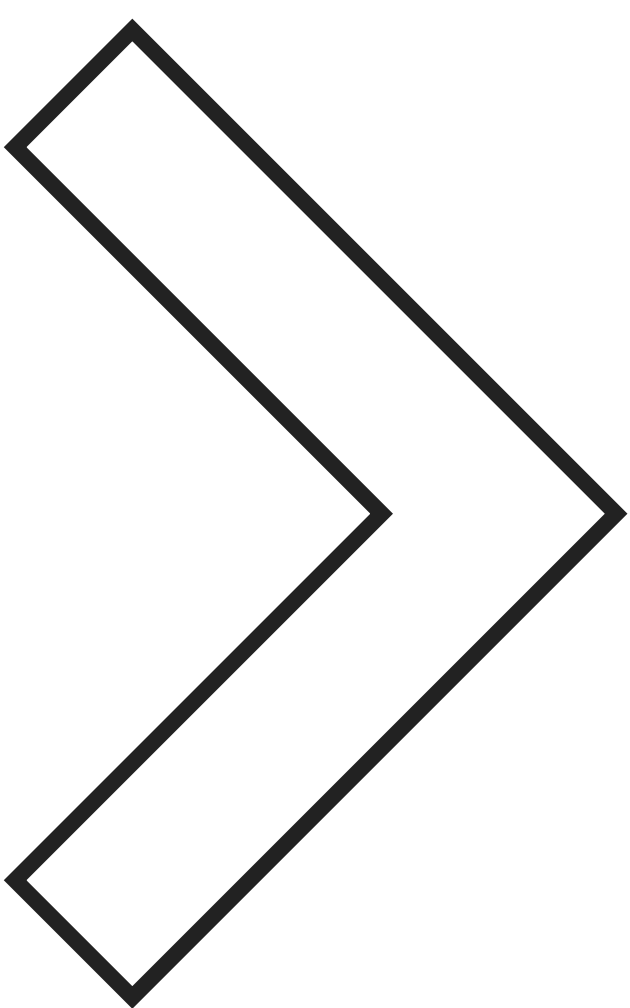
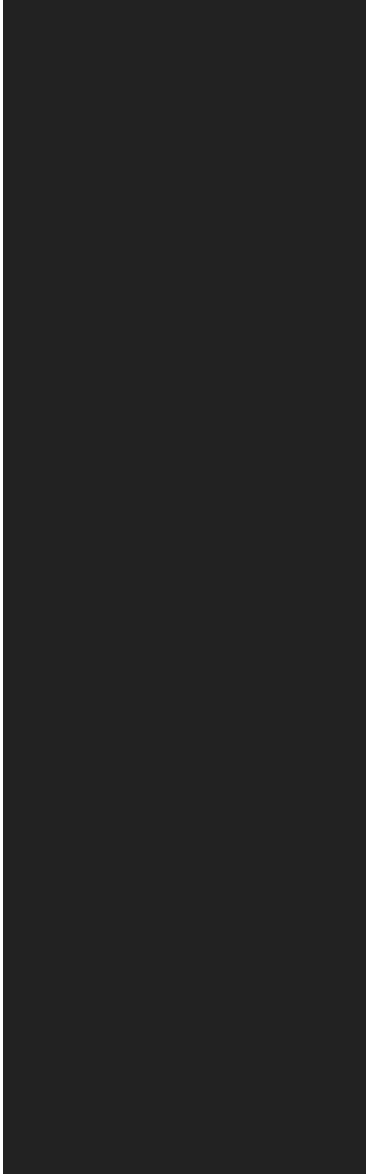
Ionic, le survivant 38

CONCLUSION 40

Comment faire
le bon choix ? 41

Take away 43

Accompagnement 44



O

r

t

u

v

LE DÉBUT D'UNE NOUVELLE ÈRE

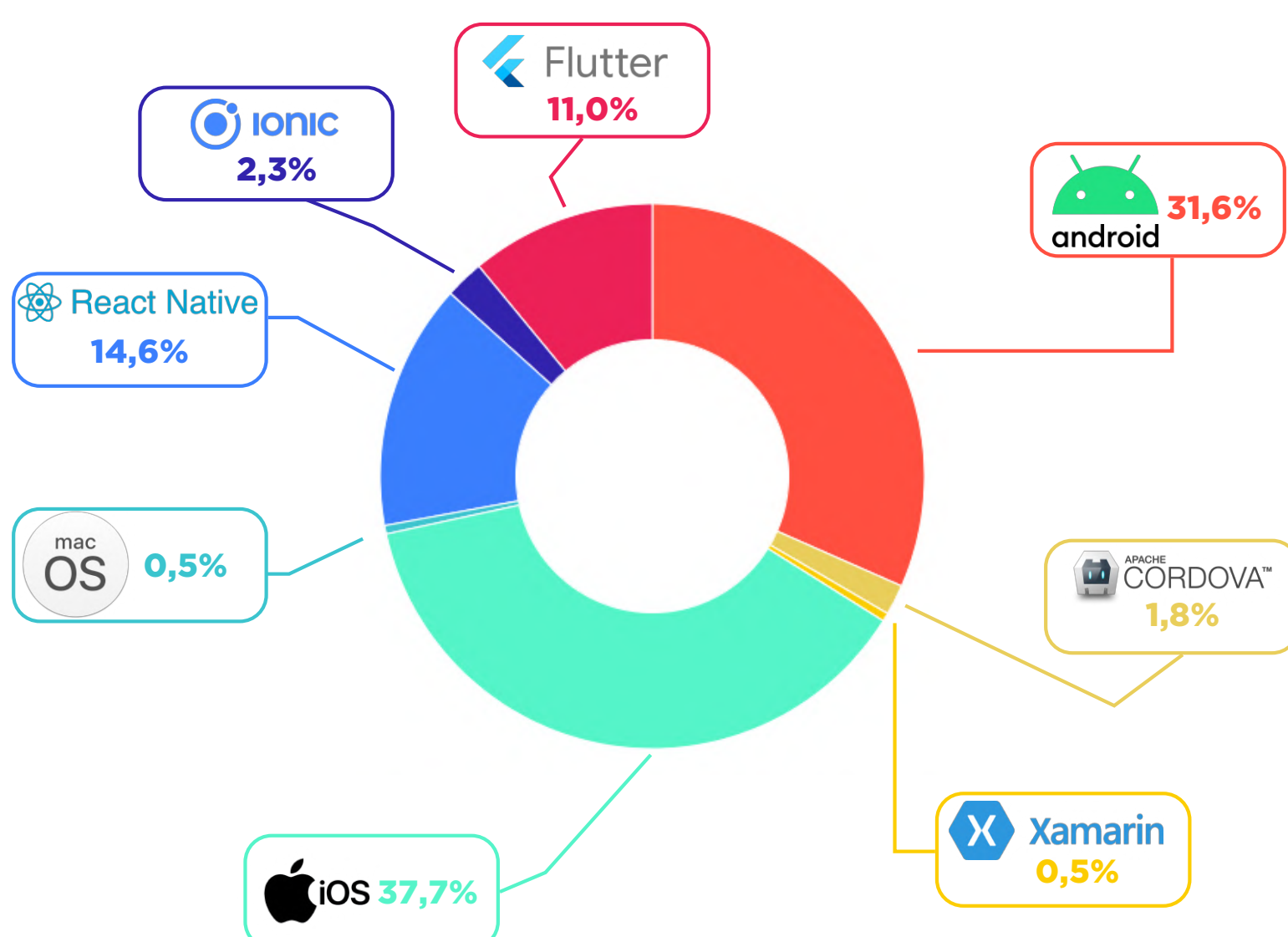
2019 - 2022. Deux ans depuis la première édition de ce guide. Entre les deux, une pandémie qui a à la fois confirmé la nécessité pour beaucoup d'acteurs de disposer d'applications mobiles, notamment pour faciliter les interactions à distance et sans contact. Mais également accéléré le besoin, pour les plus avancés, d'industrialiser et de mutualiser leur plateforme digitale au sein d'un écosystème de plus en plus développé.

Les tendances sur les langages et framework de développement d'applications mobiles, perceptibles en 2020, se précisent et viennent bouleverser la hiérarchie établie. Ce guide actualisé intègre, à son niveau, des illustrations sur ces nouvelles dynamiques.

Son objectif, lui, reste inchangé, vous donnez les clefs pour choisir la technologie de développement de vos applications mobiles de façon éclairée. Ce choix structurant conditionne la performance, le coût mais surtout la pérennité de vos applications. S'il n'est pas engageant à vie, il doit rester valable pour les 4 à 5 prochaines années.

DES TENDANCES QUI SE CONFIRMENT

L'entreprise Bitrise publie chaque année son panorama des technologies ⁽¹⁾ utilisées pour générer des applications iOS et Android via sa plateforme.



Répartition des builds par technologie sur Bitrise, 2021

Source : Bitrise, janvier 2021

¹ Bitrise, State of App Development in 2019, 2020, 2021

La comparaison de ces chiffres avec la version 2020 présentée lors de la première édition de notre guide permet d'identifier plusieurs grandes tendances :

- + Comme en 2020, les technologies natives iOS (Swift) et Android (Kotlin) restent largement majoritaires. La tendance est cependant très légèrement à la baisse, avec un recul d'environ - 1,4 % de part de marché pour les langages natifs Android et iOS.
- + Du côté des solutions hybrides, la hiérarchie est en train d'être bousculée :
 - + Flutter est la technologie avec la plus forte croissance : +45 % en 3 ans, coude à coude avec React Native qui reste stable à 14,6 % de part de marché. Cette croissance est d'autant plus forte qu'il faut considérer l'inertie des données fournies par Bitrise. Une application mobile a une durée de vie moyenne de 4 à 5 ans, c'est donc uniquement une fraction de l'ordre de 20% du parc qui se renouvelle chaque année.
 - + Les solutions historiques Xamarin (- 75 %), Ionic (- 38 %) sont en perte de vitesse. Si Xamarin (remplacé par .Net Maui depuis fin 2021) semble voué à disparaître rapidement, Ionic et Cordova ont encore leurs inconditionnels et se maintiennent à 2 % de part de marché.

	2019	2020	2021	Variation '20/'22
Android	31,0%	30,0%	31,6%	2%
Cordova	1,6%	2,0%	1,8%	13%
Xamarin	2,0%	1,0%	0,5%	-75%
iOS	39,3%	40,0%	37,7%	-4%
MacOS	0,8%	0,7%	0,5%	-38%
React Native	14,1%	13,0%	14,6%	4%
Ionic	3,7%	3,0%	2,3%	-38%
Flutter	7,6%	9,0%	11,0%	45%

APPLICATIONS NATIVES, COMPILÉES, HYBRIDES ET WEB

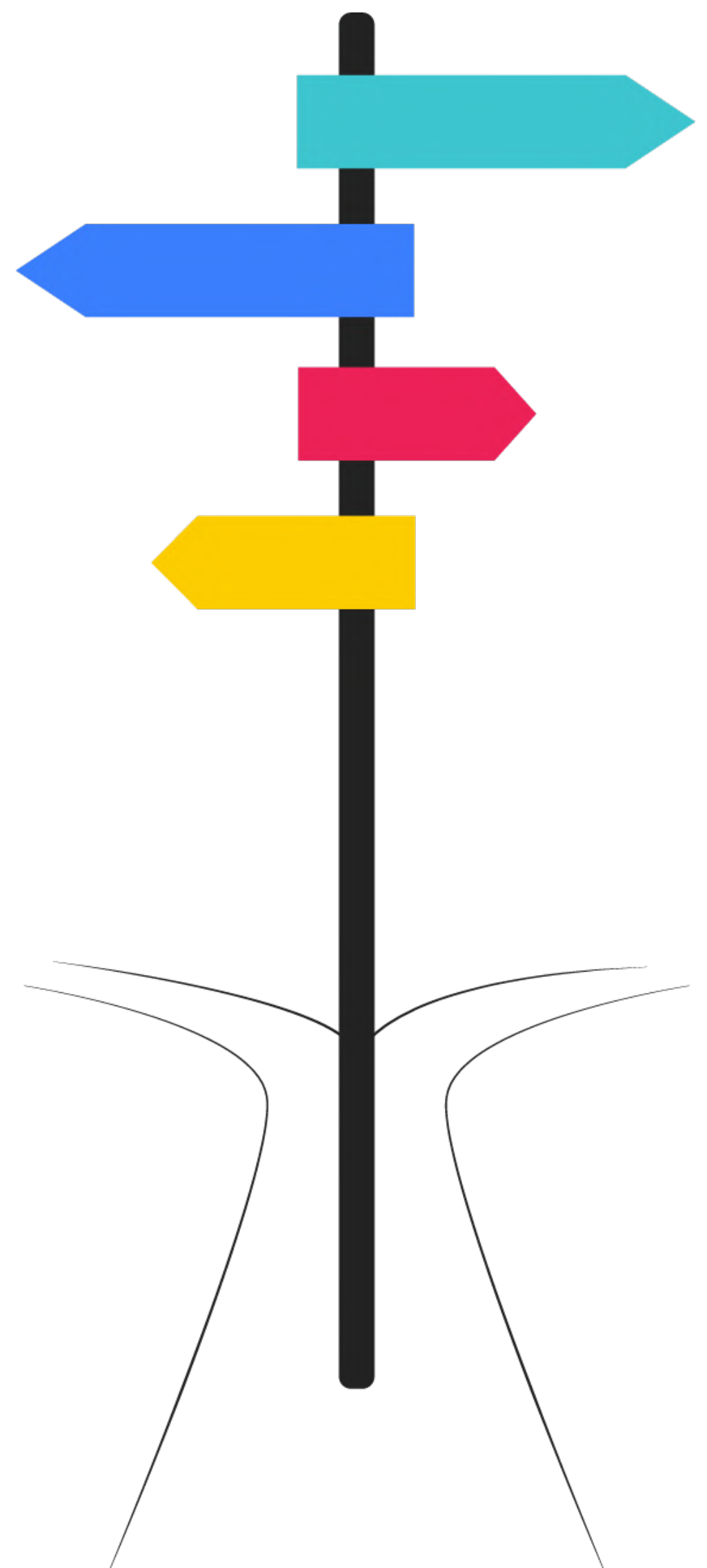
Schématiquement, les langages et technologies de développement peuvent être classés en quatre grandes catégories :

+ Les **langages de développement des systèmes d'exploitation**, dits « natifs » (langages compilés et par nature, spécifiques à chaque plateforme) permettant le développement d'applications dites elles aussi « natives » (Kotlin ou Java pour Android chez Google, Swift ou Objective-C pour iOS chez Apple).

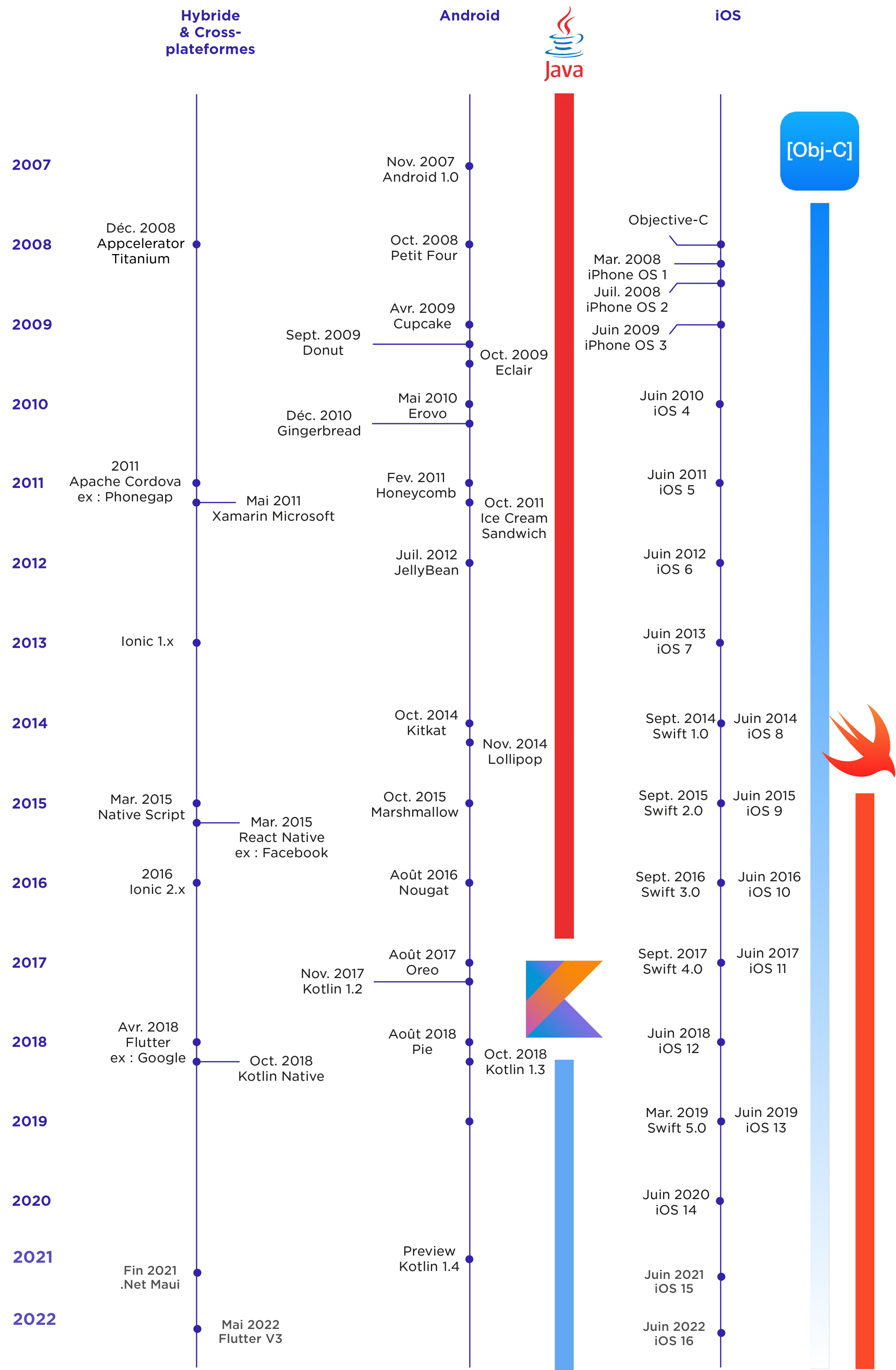
+ Les **récents langages ou solutions de développement cross-plateformes** permettant de générer du code compilé, adapté à chaque plateforme (Flutter, Kotlin/Native).

+ Les **frameworks et bibliothèques Javascript orientés mobiles** permettant le développement d'applications dites « hybrides » (React Native, Vue-Nativescript).

+ Enfin, pionnières du cross-plateforme, les **technologies** permettant d'**encapsuler des pages web** dans une **application mobile** (Ionic, Webviews).



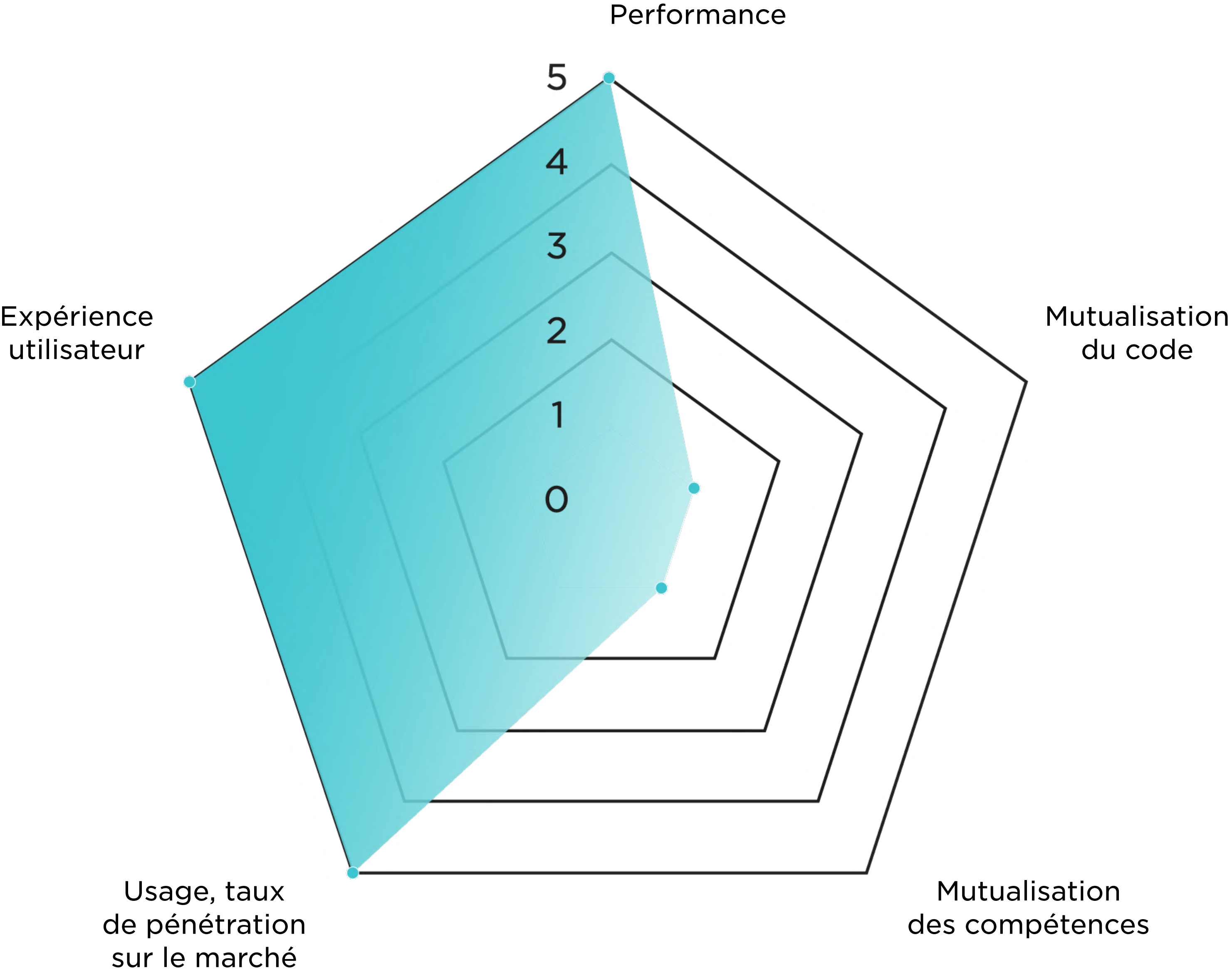
Historique des technologies mobiles



Les applications natives pour une expérience sans couture



| NATIF



Une application native est écrite dans un langage de développement principalement dédié au monde du mobile ⁽²⁾. Les langages de développement natifs sont spécifiques à chaque système d'exploitation : Objective-C ou Swift pour iOS, Java ou Kotlin pour Android.

Pour disposer d'une application native fonctionnant sous iOS et sous Android, il faut donc développer deux fois la même application en utilisant deux langages de programmation.

Plus coûteux lors de la phase de build, les applications natives, si elles sont développées dans les règles de l'art, ont l'avantage de fournir une expérience utilisateur optimale et des performances inégalables par une application hybride. Écrite en langage natif, l'application peut en effet utiliser les SDK mis à disposition par Apple et Google et exploiter pleinement les fonctionnalités des terminaux sur lesquels le système d'exploitation est installé. L'utilisation de ces SDK garantit également le respect des standards de navigation sur mobile et une gestion optimale des nombreux formats de terminaux disponibles sur le marché.



Si l'utilisation de ces composants standards simplifie le développement, il est possible de créer des éléments de navigation et d'interface sur-mesure. De manière schématique, plus le design des écrans s'appuie sur des composants “*custom*”, plus le temps de développement sera long et la maintenance complexe.

Quel que soit le type de composants utilisés, une fois écrite, l'application est compilée dans un format binaire directement interprétable par le système d'exploitation. Le fichier obtenu (.apk ou .aab pour Android et .ipa pour iOS) est ainsi optimisé pour le terminal.

À l'inverse, nous le verrons, la plupart des solutions hybrides ne génèrent pas de code binaire mais un code interprété. Ce code interprété doit être traduit lors de chaque exécution, mobilisant une partie de la puissance de calcul et dégradant ainsi les performances.

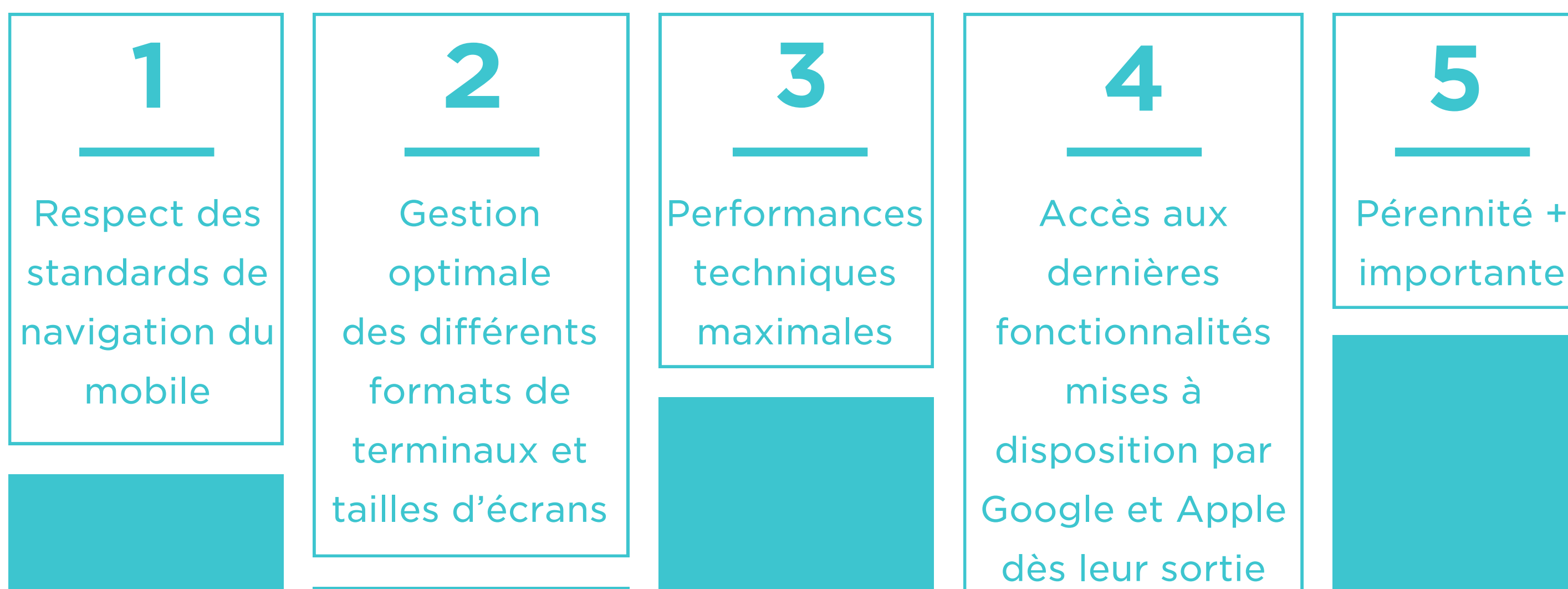
² À l'exception de Java et plus marginalement de Swift. Swift est aussi disponible sous Linux, Windows et AWS Lambda et permet par exemple de faire des API côté serveur avec les SDK Vapor, Kitura ou Perfect.

Capitalisant sur leur couplage fort et direct avec le système d'exploitation, les applications natives bénéficient donc de toute la puissance du terminal. Elles permettent également d'accéder directement aux capteurs des smartphones (accéléromètre, gyroscope, caméra, etc.) et facilitent l'utilisation de la mise en cache et le stockage des données.

Enfin, avec un langage natif, le développeur dispose de toute la palette des outils et librairies mises à disposition par Apple et Google. Côté librairie, il sera par exemple possible d'intégrer dans son application de la réalité augmentée en utilisant AR Kit et AR Core.

Dans un domaine en constante évolution, la durée de vie d'une application native sera supérieure à celle des applications dites hybrides. Certaines applications natives développées il y a 4 ou 5 ans proposent une expérience toujours d'actualité. Si la mise de départ est plus importante, le TCO (*Total Cost of Ownership*) d'une application native peut ainsi, sur l'ensemble du cycle de vie, se révéler inférieur à celui d'une application hybride.

En synthèse, le développement d'une application native, sous réserve de respect des bonnes pratiques de développement, garantit :



Mais, à l'opposé, la multiplication par deux des coûts de développement de l'application et les difficultés de recrutement des développeurs Swift et Kotlin peuvent être des freins au développement d'applications natives et peuvent justifier l'étude de solutions alternatives.

Focus - SwiftUI, le nouveau compagnon de route de Swift

En 2019 lors de sa conférence annuelle WWDC, Apple avait annoncé, conjointement à la sortie d'iOS 13, l'arrivée de SwiftUI.

SwiftUI, comme son nom l'indique, est basé sur le langage Swift. Schématiquement il permet de gérer les interfaces et composants graphiques des applications mobiles.

Successeur d'UIKit (basé lui sur Objective-C) SwiftUI est compatible uniquement à partir de la version 13 d'iOS.

Cette compatibilité n'est plus un frein à son adoption aujourd'hui, une très large majorité des possesseurs d'iphones ayant migré vers iOS 14 (14%) et iOS15 (82%) ⁽³⁾.

SwiftUI - aujourd'hui dans sa version 3 et annoncé en version 4 avec la sortie d'iOS 16 - est de plus en plus intégré par Apple dans les applications fournies par défaut (Livres, Plans, Notes, Météo, Musique, Podcasts) ⁽⁴⁾.

Si une montée en compétence sur SwiftUI est nécessaire, son utilisation facilite le développement et la maintenance d'applications iOS. De nouvelles fonctionnalités comme SwiftUI Charts permettant de générer des graphiques, viennent régulièrement enrichir ce framework, l'imposant de facto dans l'écosystème Apple.

³ iOS usage, Apple

⁴ Apple's use of Swift and SwiftUI in iOS 15, Dec 2021

Focus - Un partenariat fructueux entre Google et JetBrains à l'origine d'Android Studio, Kotlin, JetpackCompose

JetBrains est une société Tchèque (voir zoom “JetBrain choisit son camp”) qui conçoit des outils à destination des équipes de développement.

Avec une forte spécialisation sur l'univers Android, JetBrains travaille de concert avec Google sur plusieurs produits.

Historiquement, JetBrains est connu pour être l'éditeur d'IntelliJ IDEA, l'environnement de développement intégré (IDE) Java sur lequel est basé Android Studio (l'IDE officiel de Google pour Android).

Cette société est également à l'origine du langage de développement Kotlin. Adoubé par Google lors de la conférence Google I/O 2019, Kotlin est désormais le langage recommandé par Google pour les développeurs d'applications Android et se généralise progressivement (en 2019 environ 60% des applications Android du top 1 000 contenaient du code Kotlin ⁽⁵⁾).

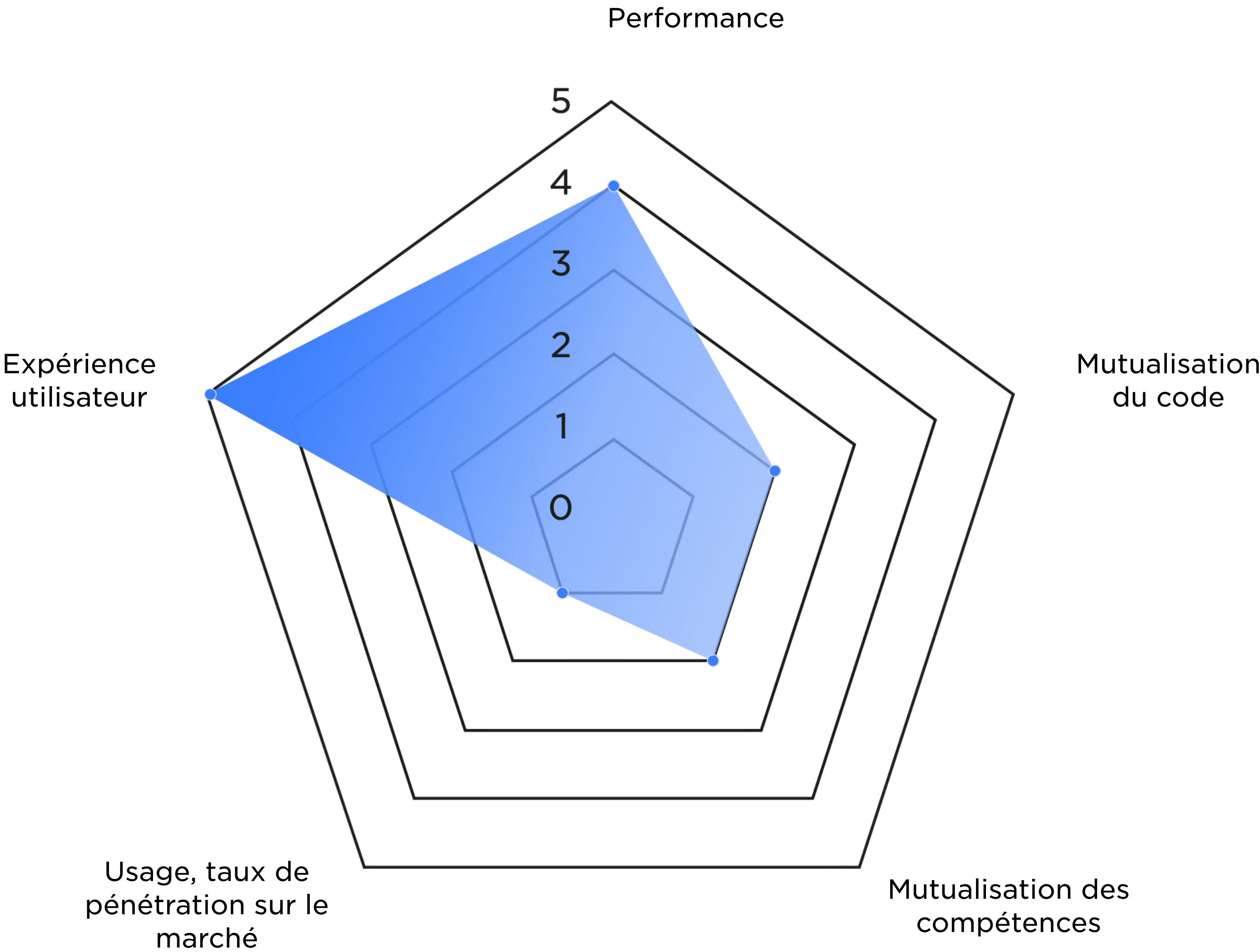
Enfin, co-développé par Google et JetBrains, Jetpack Compose, l'équivalent de SwiftUI pour Android, est depuis juillet 2021 disponible en version 1. Utilisable en production couplé à Kotlin, il vise tout comme son équivalent iOS à simplifier le travail des développeurs et permet de s'affranchir du formalisme XML jusqu'alors nécessaire pour composer les vues d'une application Android.

Si la collaboration entre Google et JetBrains est productive sur les technologies natives, leurs chemins semblent à ce stade diverger sur les technologies cross-plateformes avec, côté Google la promotion de Flutter et côté JetBrains, celle de Kotlin Multiplatform.

La revanche des applications cross- plateformes



| KOTLIN MULTIPLATFORM



Le partage de code Android vers iOS avec Kotlin Multiplatform Mobile : la mutualisation sans effort ?

Si Kotlin permet de développer des applications Android, il est aussi possible depuis la version 1.3 de Kotlin (sortie en janvier 2019) avec Kotlin Multiplatform de compiler le code écrit en Kotlin dans différents formats : Kotlin/JVM pour Android mais aussi Kotlin/Native pour iOS ⁽⁶⁾. Le code compilé en Kotlin/Native peut être intégré dans une application iOS écrite en Objective-C ou Swift. Il est donc possible de réutiliser du code Android pour une application iOS sans concession sur la performance puisque le code est compilé comme avec Objective-C ou Swift.

Autre avantage de Kotlin Multiplatform : la prise de risque est minimale. Ce sont bien des parties du code de l'application Android, sous réserve d'être correctement structurées, qui sont extraites et réutilisées pour l'application iOS.

Attention cependant, 100% du code Android n'est pas réutilisable. Nous estimons la part de code mutualisable entre Android et iOS comprise entre 30% et 50%. Pour mieux comprendre la part mutualisable de code, il faut faire un petit détour par les paradigmes d'architecture.

⁶ A noter qu'avec Kotlin Multiplatform le code peut aussi être partagé avec un site web écrit, par exemple, en Javascript étendant les possibilités de convergence au monde du web.

Un paradigme d'architecture permet de structurer le code en l'organisant en différentes couches. Chacune des couches ayant un rôle précis. Dans le pattern appelé MVC (Modèle, Vue, Contrôleur), par exemple, la couche Modèle va décrire le modèle de données, la couche Vue va gérer l'interface et l'affichage des composants alors que la couche Contrôleur va, elle, faire le lien entre la Vue et le Modèle. D'autres paradigmes d'architecture existent notamment MVVM, MVP, VIPER. Pour Android, le paradigme de référence est MVVM. Côté iOS, historiquement structurées en MVC notamment avec UIKit, l'arrivée de SwiftUI permet de s'adapter à tous les paradigmes y compris le modèle MVVM.

Kotlin Multiplatform permet de mutualiser les « couches basses » de l'application (les couches Modèle et Contrôleur) mais la couche Vue, elle, reste spécifique à chaque application. Pour iOS, cette couche sera par exemple basée sur SwiftUI.

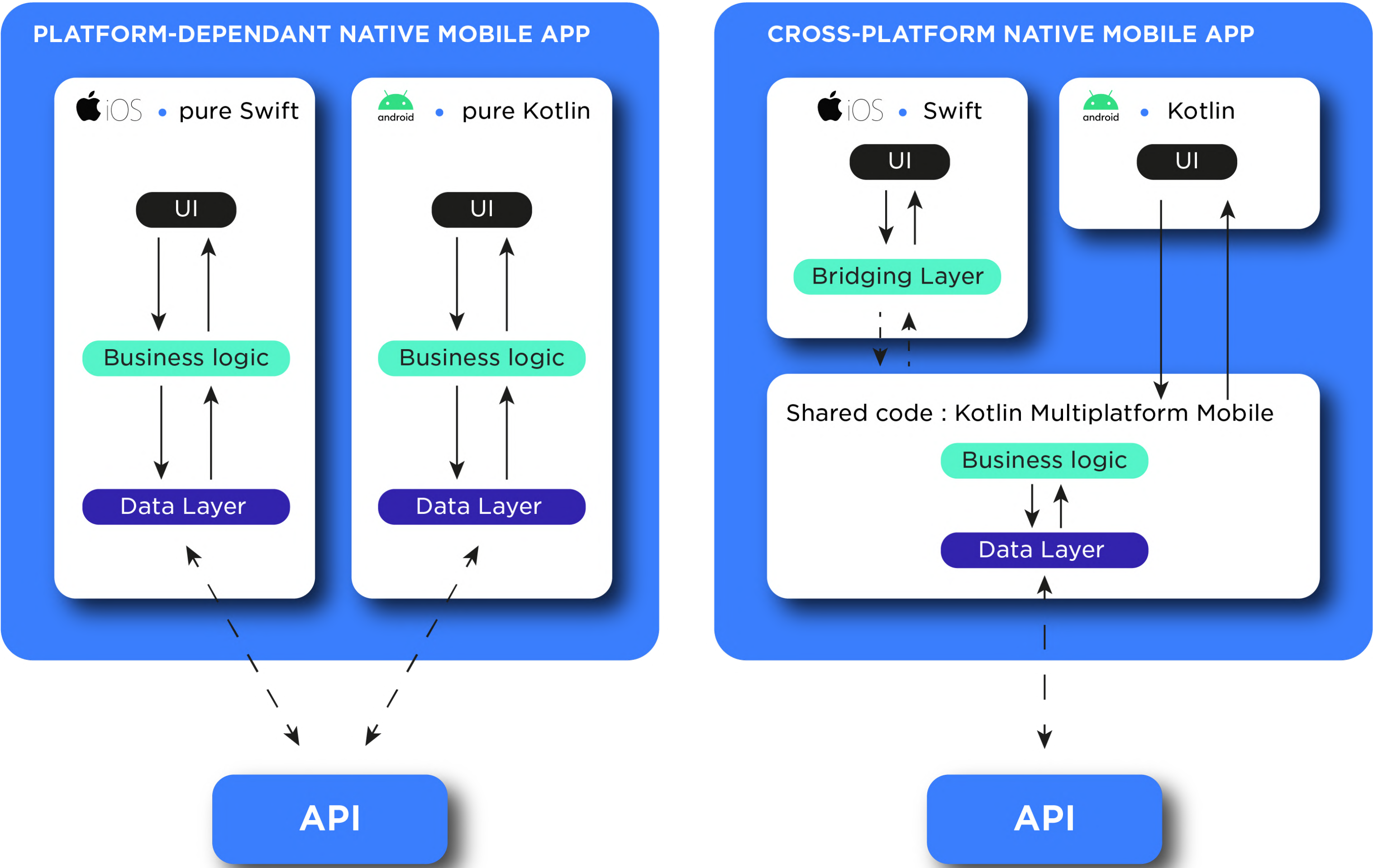
Si l'intégration de briques Kotlin/Native dans une application iOS n'impose pas en soi de pattern d'architecture, l'utilisation du pattern MVVM (recommandé pour le développement d'application Android en Kotlin et pour le développement d'application iOS avec SwiftUI) est préférable.

Malgré cette mutualisation partielle, la promesse de Kotlin Multiplatform reste alléchante et prometteuse. Mais, si l'adoption sur le marché de Kotlin est incontestable, celle de Kotlin/Native reste encore incertaine. Cette technologie est encore récente (janvier 2019) et, même si quelques sociétés telles que Square ⁽⁷⁾ sautent le pas, il existe peu de références majeures sur le marché.

Enfin, l'utilisation de Kotlin/Native nécessite aussi de repenser le déroulé d'un projet. Afin de pouvoir inclure des packages Kotlin/Native dans une application iOS, il est nécessaire de décrire un « contrat d'interface » permettant de partager un même vocabulaire entre les deux applications.

⁷ Square Avril 2019 - The convergence of iOS and Android development

Architecture schématique d’une application native
et d’une application cross-plateforme basée sur Kotlin Multiplatform



JetBrains choisit son camp

Les fondateurs de JetBrains, la société à l'origine du langage de développement Kotlin, sont d'origine russe.

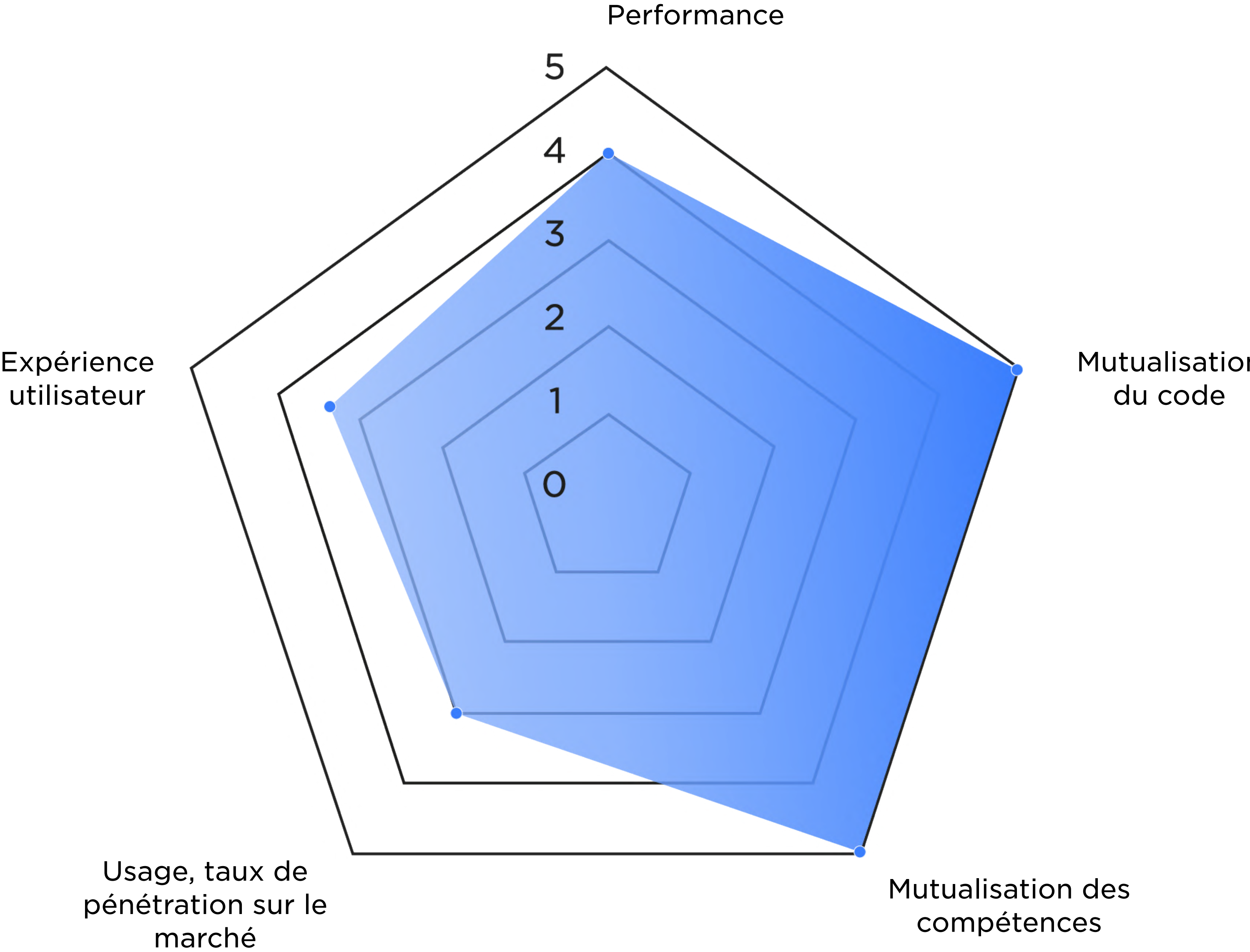
Historiquement, et même si le siège social de la société est basé à Prague en République Tchèque, JetBrains disposait donc d'un lien fort avec la Russie et avec notamment des bureaux de R&D à Saint Pétersbourg.

La société a rapidement clarifié sa position contre la guerre en Ukraine, tout d'abord au travers de tweets, puis annonçant officiellement en mars 2022 ⁽⁸⁾ la suspension de toutes ses activités commerciales et R&D en Russie et en Biélorussie.

Même si JetBrains annonce que des employés localisés en Russie ont déjà, pour certains, changé de pays, cette décision a bien évidemment eu un impact important. Les fondateurs parlent de la décision la plus importante qu'ils aient prise jusqu'à aujourd'hui.

⁸ *JetBrains' Statement on Ukraine - Mars 2022*

| FLUTTER



Flutter : le pari gagné de Google ?

Sorti en décembre 2018, Flutter est un framework de développement permettant le développement unique d'interfaces mobile et web avec le même code.

Flutter est basé sur le langage Dart créé par Google. Ce nouveau langage de développement, conçu à la base pour remplacer le Javascript, peut désormais être compilé en Javascript mais aussi en langage machine ⁽⁹⁾. C'est donc un langage universel qui peut être utilisé côté front comme côté serveur.

En s'appuyant sur Dart, Flutter se positionne au même niveau que Kotlin Multiplatform en termes de performances, le code produit étant compilé pour la plateforme ciblée.

Flutter s'appuie sur la notion de widget : ⁽¹⁰⁾ une application ou un site est créé en agencant un ensemble de widgets Flutter. Un widget définit le comportement d'un objet, son interface, etc. La notion de widget unifie donc l'ensemble des couches vue, contrôleur, etc. qui sont séparées dans d'autres frameworks.

Certains widgets Flutter respectent les guidelines Material Design de Google (des widgets estampillés MaterialUI sont fournis ⁽¹¹⁾) et d'autres proposent des interfaces proches des guidelines iOS ⁽¹²⁾. Pour autant, ce sont bien des composants Flutter qui sont utilisés. Une fois générée, l'application développée sur Flutter aura donc un rendu identique sur iOS et Android (sauf à spécifier dans le code l'usage de composants différents selon la plateforme).

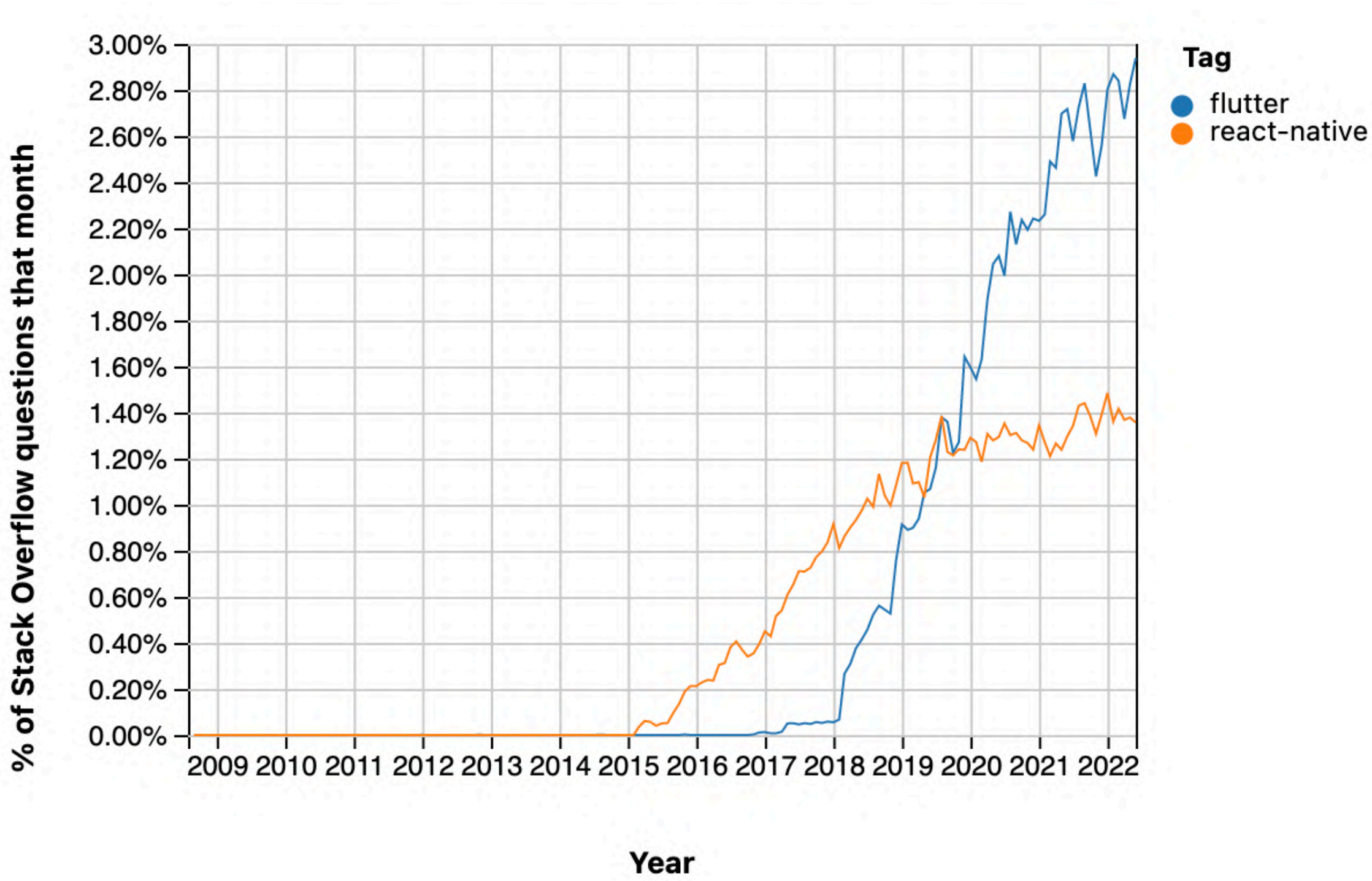
Déjà palpable en 2020, l'engouement grandissant autour de Flutter s'accélère chez les développeurs. La prise en main de cette nouvelle stack Flutter (et en sous-jacent celle de Dart) est au RDV, comme illustré sur les deux graphiques suivants.

⁹ Dart Platform

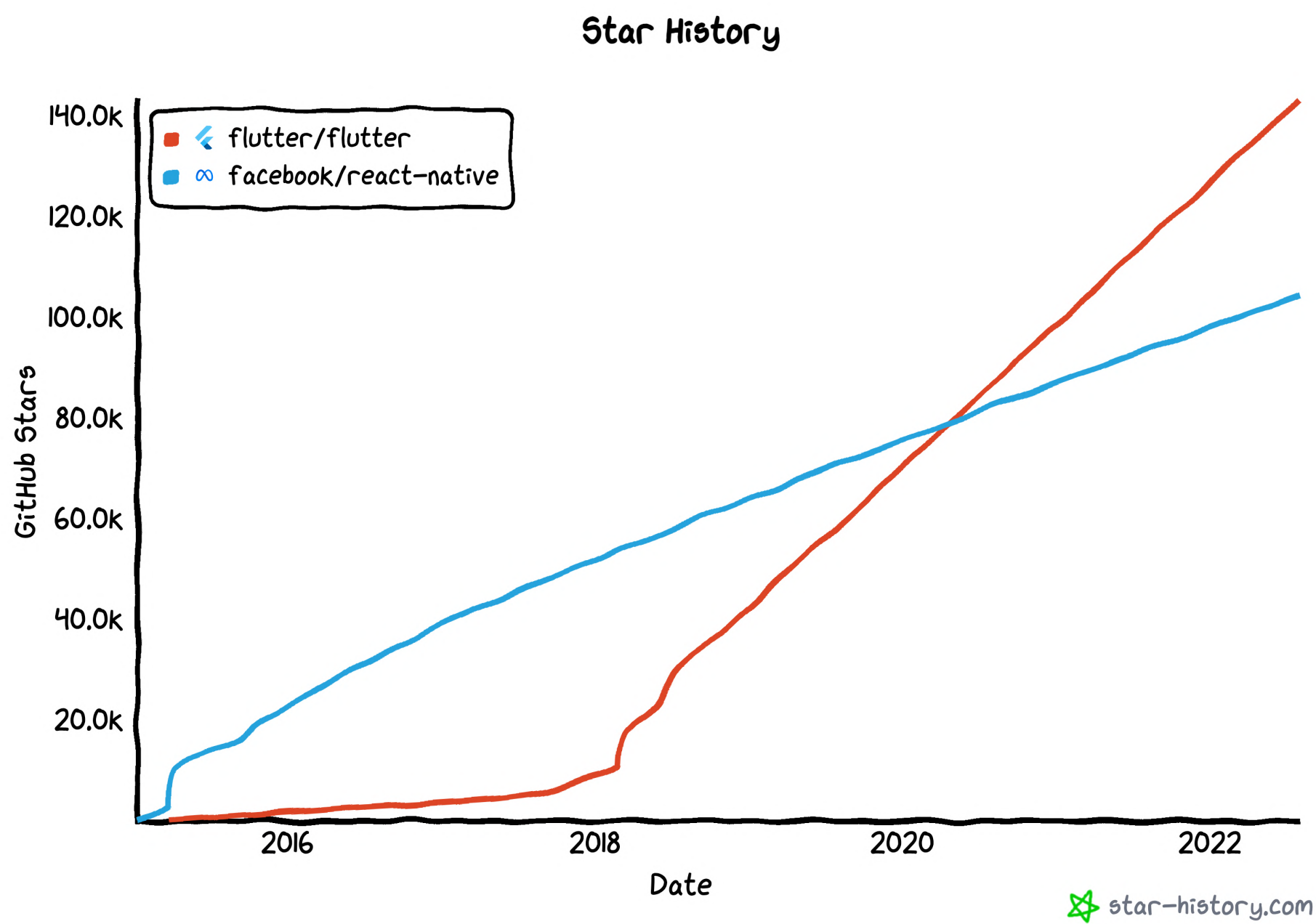
¹⁰ Catalogue de Widget de Flutter

¹¹ Material Components widgets

¹² Cupertino (iOS-style) widgets



% de questions sur stackoverflow



Popularité de Flutter et React Native sur GitHub

Si initialement l'avenir de Flutter semblait conditionné au développement et au succès de Fuchsia (le nouvel OS de Google en préparation depuis plusieurs années dont l'interface est basée sur Flutter), son avenir semble dès à présent garanti, alors que Fuchsia lui reste encore confidentiel et en construction.

L'utilisation de Flutter se démocratise également sur les applications grand public. En France, la refonte et le lancement de l'application *Mon Compte* illustre cette tendance de fond du marché. Sur ce cas précis, le choix de Flutter n'a pas été remis en cause et ne semble pas être à l'origine des difficultés de prise en main de la nouvelle version par les utilisateurs.

Un autre point d'attention à considérer lors du choix de Flutter est la compatibilité avec des solutions tierces. Pour illustrer cette problématique, l'intégration, par exemple, d'une solution de tracking dans une application nécessite (quel que soit le langage de développement utilisé pour développer l'application Swift, Kotlin, React Native, etc.) l'ajout d'un SDK. Ce SDK est mis à disposition par l'éditeur. Si les SDK Kotlin et Swift sont aujourd'hui existants et mis à disposition par tous les bons éditeurs disposant de solutions adaptées au mobile, les SDK compatibles Flutter sont encore parfois manquants et peuvent obliger à engager une discussion avec l'éditeur ou choisir une solution alternative. Le site officiel de Google, <https://pub.dev>, recense les packages et SDK compatibles Flutter et Dart.

D'autres exemples d'applications réalisées en Flutter sont mises en avant par Google sur le site de Flutter

👉 <https://flutter.dev/showcase>

Flutter double la mise sur le web ?

Si Flutter prend des parts de marché sur le développement des applications, Google a étendu son périmètre bien au-delà.

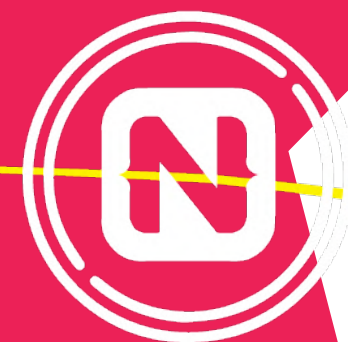
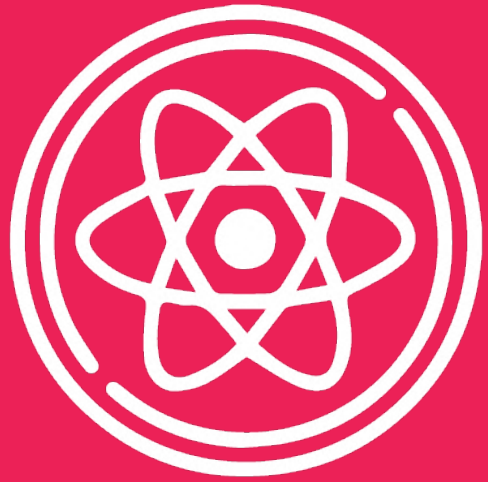
Compatible avec les navigateurs web et avec Windows depuis sa version 2, la version 3 de Flutter permet maintenant de générer de façon stable des applications pour MacOS ainsi que pour les distributions Linux.

L'apparition de technologies qui permettront, dans un futur proche, d'adresser l'ensemble des plateformes semble inéluctable. Google est certainement l'acteur le plus légitime pour mettre au point cette technologie avec sa mainmise sur Android et son expérience sur Angular (qui reste l'un des frameworks Javascript les plus populaires de ces dernières années).

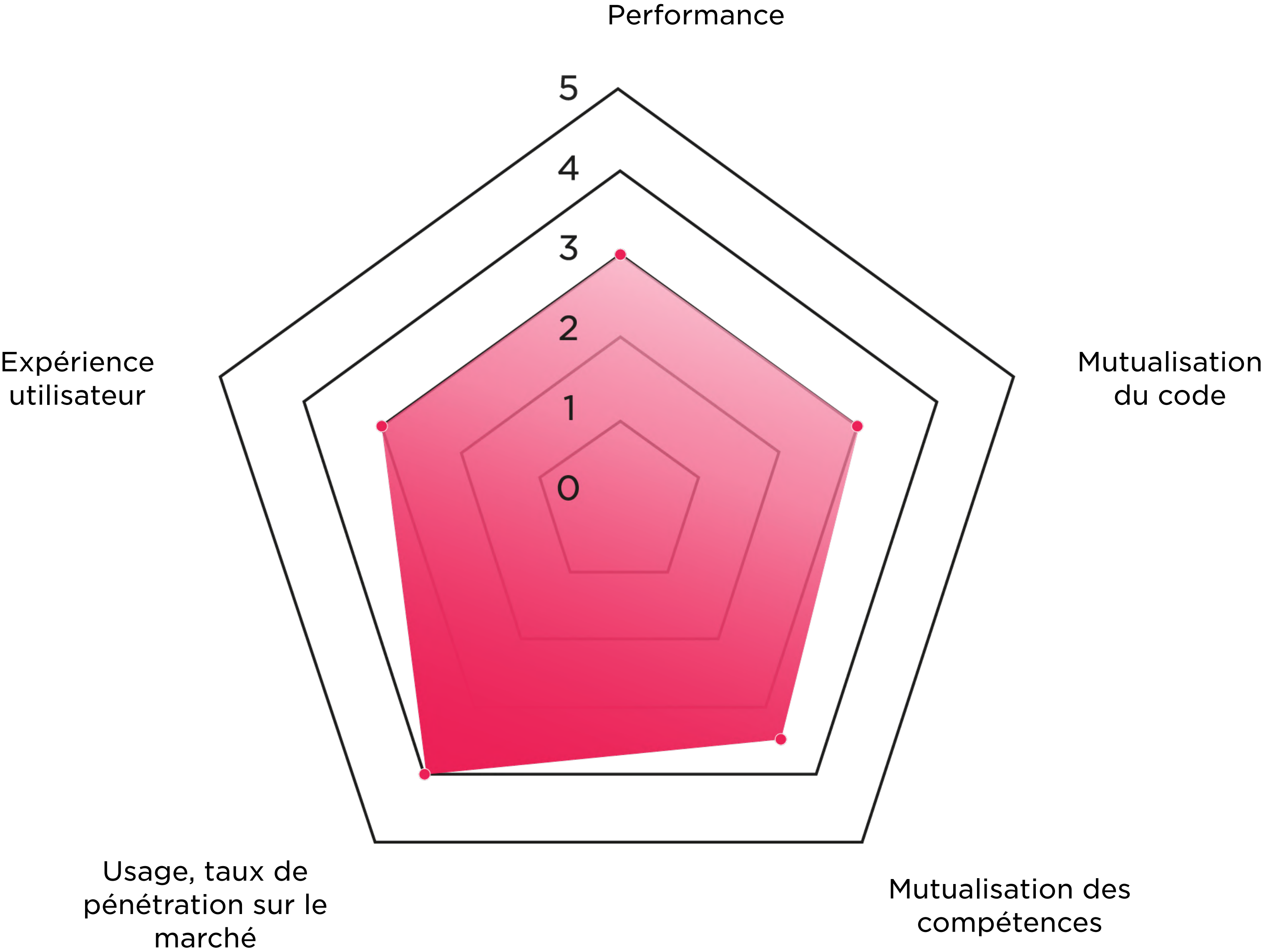
Le pari reste malgré tout très ambitieux.

Si à date l'hégémonie de Javascript et des frameworks associés (Angular, React, Vue, Svelte, etc.) sur le web est bien loin d'être remise en question (le code Flutter est lui-même "transformé" en Javascript pour être exécuté par les navigateurs), le taux de pénétration de Flutter sur le web sera à surveiller.

Le Javascript à la conquête du mobile



| REACT NATIVE



React Native : le « renouveau » de l'hybride

À la différence de Swift ou Kotlin, React Native n'est pas un langage de programmation mais un framework Javascript dédié au mobile. Créé par Facebook, la 1ère version a été publiée en 2015.

Aujourd'hui, React Native est toujours en version bêta mais a déjà largement fait ses preuves sur des milliers d'applications. React Native est utilisé par les porte-étendards de la galaxie Facebook mais aussi par Skype, Bloomberg, Pinterest, Walmart, Uber ⁽¹³⁾.

Surfant sur la vague du Javascript, base du web moderne, React Native est très populaire auprès de la communauté des développeurs web.

77% des développeurs interrogés dans le cadre de l'étude annuelle State of JS se disent intéressés par apprendre ou pour continuer à utiliser React Native⁽¹⁴⁾. React Native s'appuie sur cette communauté pour faire évoluer son framework (+ 2 000 contributeurs sur GitHub) ⁽¹⁵⁾ et enrichir son écosystème. À titre d'exemple, la bibliothèque react-native-datetimepicker fournit un composant de sélection de date. Autre exemple, react-native-camera permet d'accéder à la caméra et inclut une solution de scan de code-barres. Cet écosystème est l'une des forces de React Native.

¹³ React Native Showcase

¹⁴ State of JS 2021

¹⁵ <https://github.com/facebook/react-native>

Le développement d'une application React Native s'appuie sur la notion de composant. React Native fournit par défaut une bibliothèque de composants standards ⁽¹⁶⁾, tels que les composants Text, Image ou encore TextInput pour permettre à l'utilisateur de saisir des informations. Le rendu graphique de chaque composant peut être personnalisé. Il est également possible de développer ses propres composants en s'appuyant sur les composants unitaires fournis par le framework. Le même concept est présent chez React, le grand frère de React Native dédié au web. Attention cependant, les composants React Native sont spécifiques au mobile et ne sont pas utilisables avec React. Malgré le partage des notions et concepts entre React et React Native, les composants écrits en React Native ne sont donc pas transposables directement pour un site web ⁽¹⁷⁾. Et inversement. Une mutualisation du code sur les autres parties du projet est elle parfaitement possible (stores, routes, etc.)

Autre point limitant, à la différence des langages mobiles natifs, le code produit avec React Native n'est pas compilé mais interprété.

C'est-à-dire que lors de chaque exécution d'une action au sein de l'application, une brique va interpréter les événements provenant du système d'exploitation du terminal pour les transmettre à l'application écrite en Javascript. Idem dans le sens inverse. Cet interprète, appelé Bridge, est au cœur du fonctionnement de React Native (voir paragraphe "Une architecture technique repensée pour gagner en performance"). Écrit en langage bas niveau C/C++, il permet de faire l'interface entre le code Javascript et le système d'exploitation natif du terminal. Même optimisée, cette étape impacte les performances de l'application. Ainsi, une application hybride React Native sera moins performante, plus lourde et moins fluide qu'une application générée avec un langage compilé.

Malgré ces limites, React Native rend accessible le développement d'applications mobiles aux développeurs Javascript venant du monde du web et permet de produire, avec un code unique, des applications iOS et Android. Les coûts de développement et de maintenance d'une application mobile sont ainsi fortement réduits.

Enfin, même si React et React Native sont des frameworks distincts, les compétences de développement en Javascript peuvent être mutualisées.

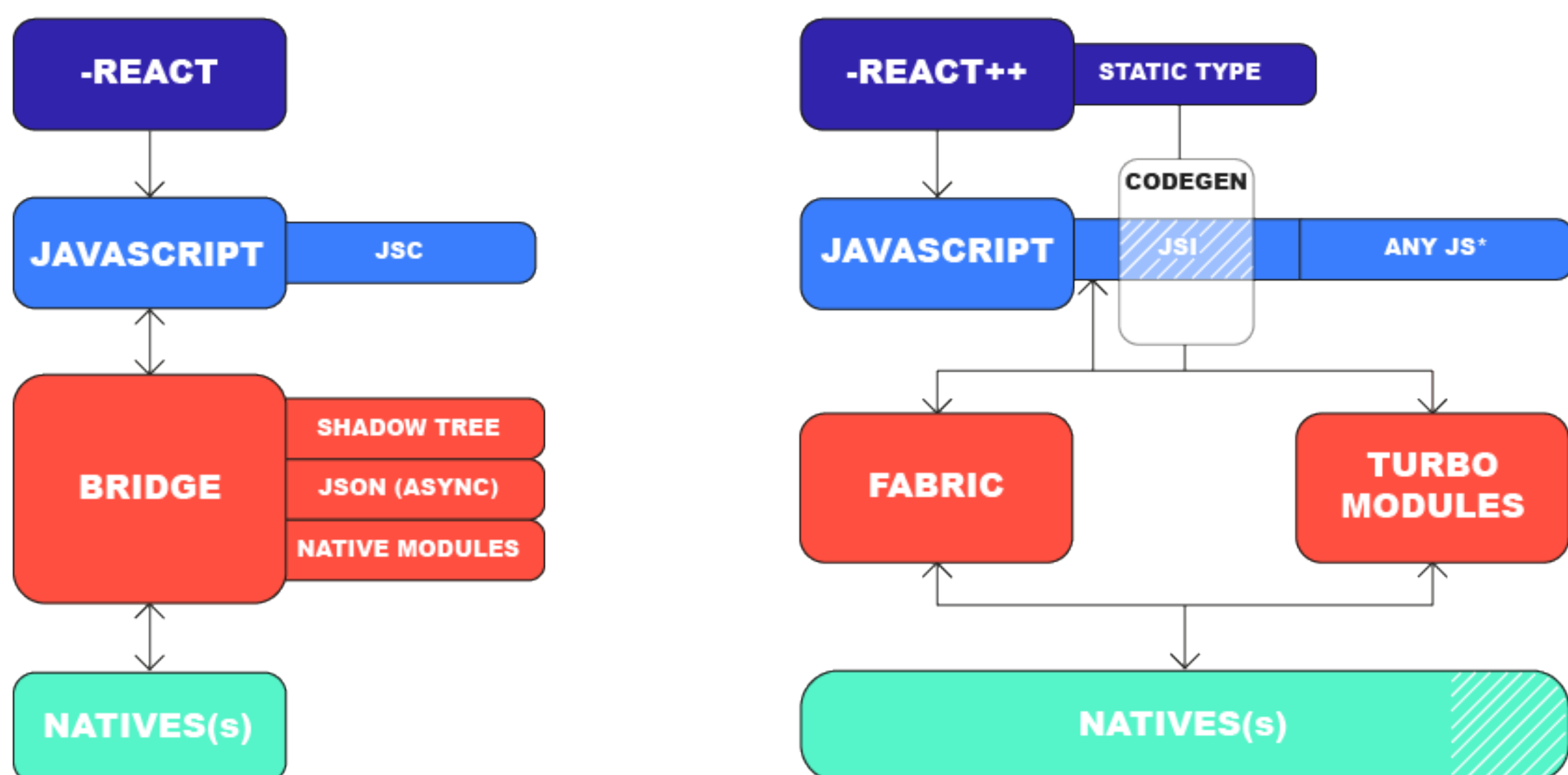
¹⁶ Liste des composants de React Native

¹⁷ A noter l'existence de la bibliothèque react-native-web qui permet de générer un site web à partir de code react-native

Une architecture technique repensée pour gagner en performance

Travail de longue haleine engagé par Facebook dès 2018, React Native bénéficie désormais d'une nouvelle architecture.

La notion de Bridge est supprimée et remplacée par l'ajout de nouvelles briques : Fabric, JSI et Turbo (voir schéma ci-dessous).



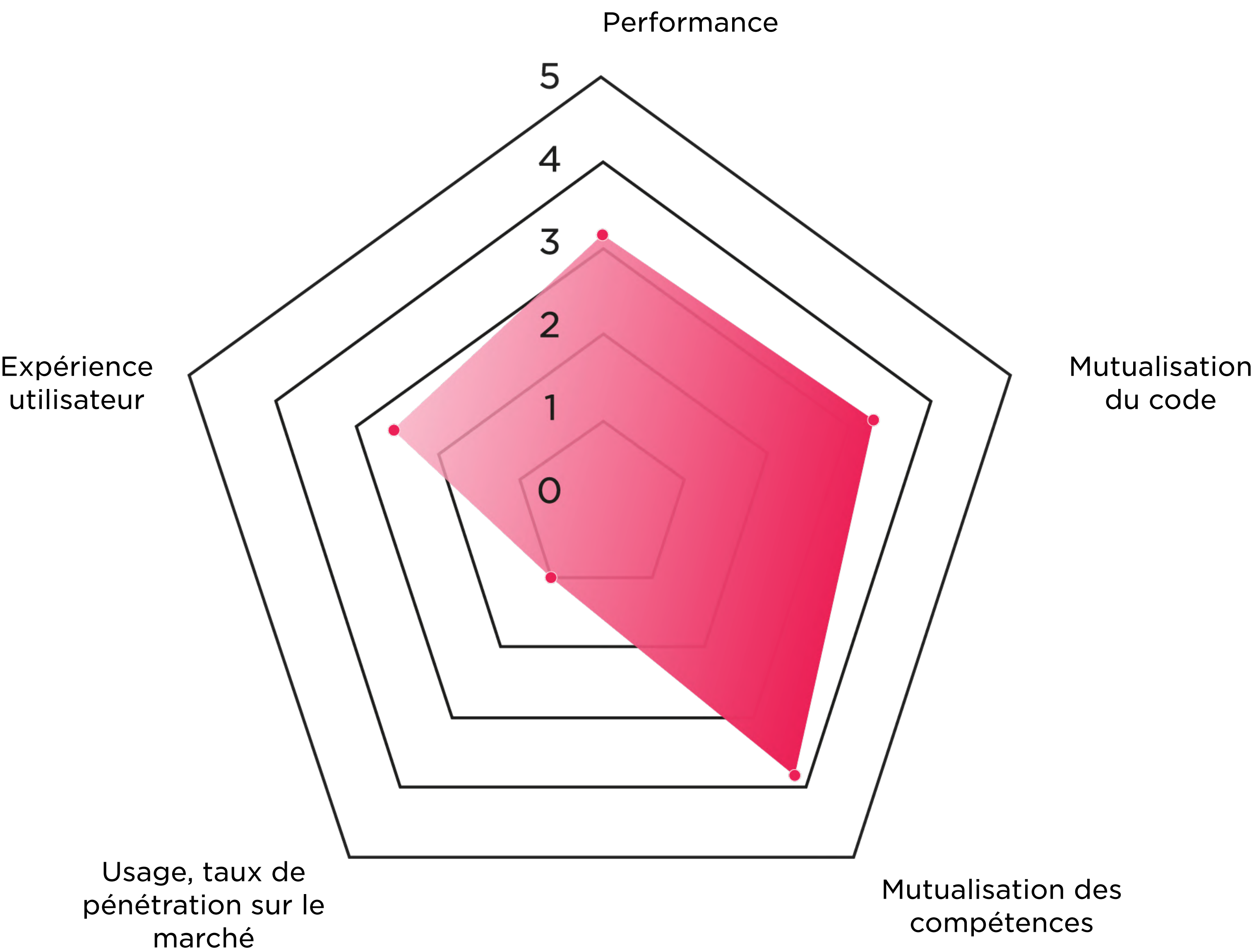
L'application Facebook a migré vers cette nouvelle architecture en juillet 2021. Depuis la version 0.68 de React Native, cette nouvelle architecture est mise à disposition de tous.

Au-delà des considérations très techniques (voir les articles mentionnés en pied de page ⁽¹⁸⁻¹⁹⁾), cette modification d'architecture induit un couplage et des échanges plus directs et synchrones entre la partie native et l'application React Native.

¹⁸ *The New React Native Architecture Explained*

¹⁹ *React Native's upcoming re-architecture - Oct 2021*

| NATIVESCRIPT



NativeScript, le parent pauvre de la convergence

Beaucoup moins populaire que React Native, les deux autres grands frameworks frontend Javascript Vue.js et Angular possèdent leurs déclinaisons mobiles avec NativeScript-Vue et Angular NativeScript.

Si React Native est basé sur JSX, une syntaxe Javascript qui ressemble au HTML, la syntaxe de NativeScript, elle, est basée sur du XML.

Concrètement, les balises utilisées en NativeScript-Vue seront différentes des balises utilisées pour Vue.js qui, elles, sont plus proches du HTML. Tout comme React et React Native, les deux codes Vue.js et NativeScript seront donc légèrement différents.

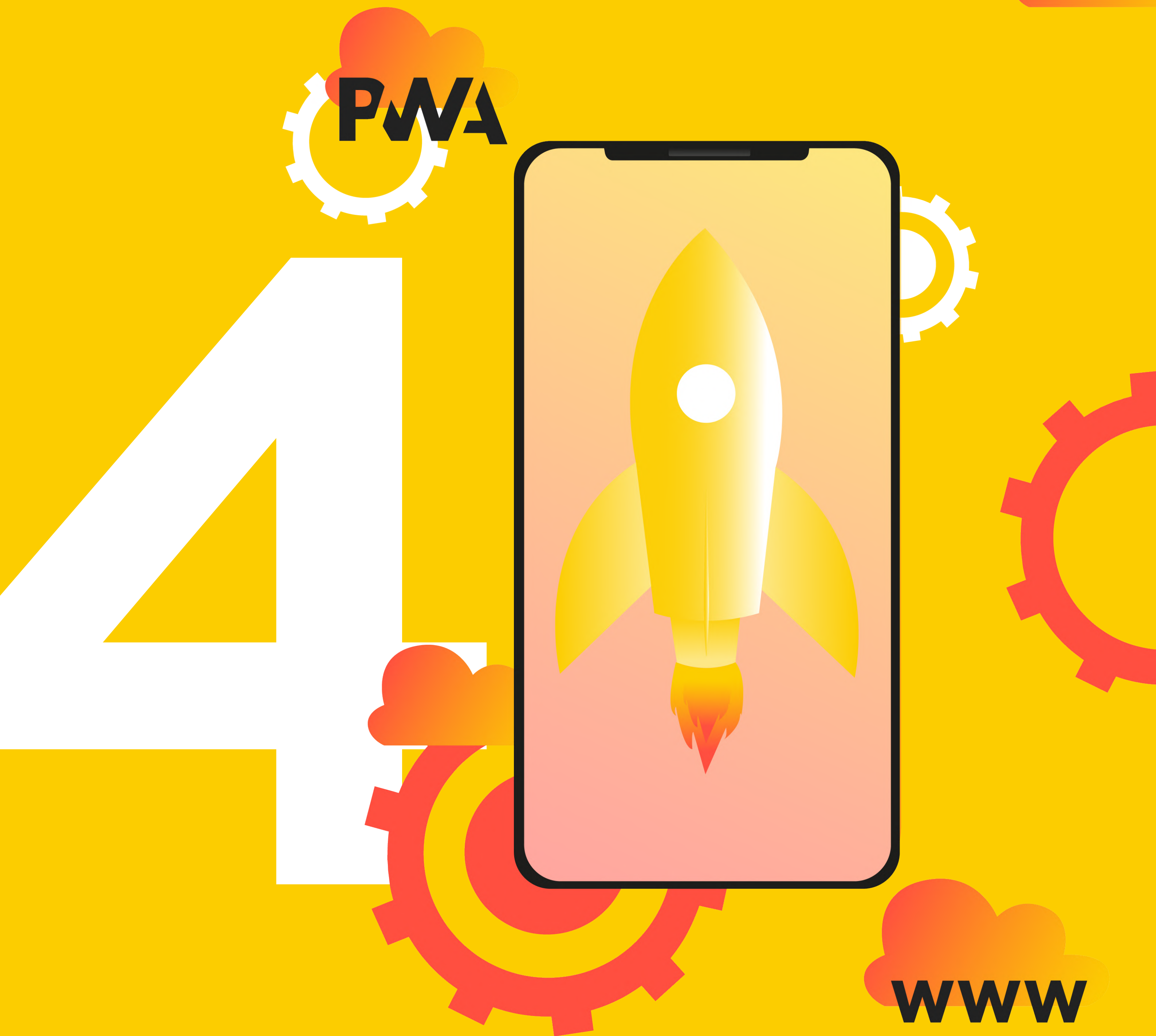
Côté Angular, le partage de code entre le web et les applications mobiles est plus simple à mettre en œuvre. On peut alors partager le même code pour le Web, iOS et Android ⁽²⁰⁾.

Si sur quelques aspects techniques la construction de NativeScript lui donne un avantage face à React Native (utilisation du moteur V8 de Google pour Android et également pour iOS depuis la version 7 de NativeScript en août 2020) ⁽²¹⁾, la popularité et l'adoption de NativeScript reste toujours très faible en comparaison de celle de React Native. Aujourd'hui, le choix de NativeScript peut se justifier pour les entreprises souhaitant développer une application cross-plateforme et disposant de compétences Angular.

²⁰ *Apps That Work Natively on the Web and Mobile - Août 2018*

²¹ *What are the key difference between ReactNative and NativeScript? - Mai 2016*

Le web encapsulé dans les applications mobiles



Les webviews, toujours d'actualité, toujours utiles

Le mécanisme de webview permet d'encapsuler une page web dans une application, tout en laissant la possibilité de masquer certaines parties (header, footer).

Ce modèle est relativement simple à mettre en œuvre. Attention cependant, une application conçue en majorité avec des webviews aura un grand risque d'être rejetée lors de sa soumission sur l'AppStore. Autre inconvénient, le contenu d'une webview ne sera pas accessible en mode hors ligne et l'écran affiché reste avant tout basé sur une page web avec un comportement parfois éloigné des standards mobiles.

Malgré tout, cette technique est parfaitement d'actualité pour des pages peu interactives telles que les FAQ ou les Conditions Générales d'Utilisation. Compatibles avec d'autres langages / techniques, les webviews sont présentes dans de nombreuses applications, même natives.

Le principal avantage est bien sûr la réutilisation du code web existant. La mutualisation entre les plateformes est ainsi maximale et l'effort minimal.

Les Progressive Web Apps (PWA) et les Trusted Web Activities (TWA) : une nouvelle incursion du web dans nos smartphones

Apparues successivement en 2015 et février 2019, les notions de Progressive Web App (PWA) puis de Trusted Web Activities (TWA) remettent au goût du jour les webviews, en rapprochant les mondes du web et du mobile, non sans générer quelques incompréhensions.

Une Progressive Web App, malgré son nom, reste avant tout un site web disposant de nouvelles fonctionnalités. Pour créer une PWA, il faut disposer d'un site web responsive et performant, y ajouter un Manifest (fichier contenant la description du site, un logo...) et surtout mettre en place des **Service Workers**.

Les Service Workers peuvent être vus comme une gare de routage. Exécutés au niveau des navigateurs web (Firefox, Chrome, Safari, etc.), ils permettent, par exemple, d'afficher une page web même hors connexion (si cette page a déjà été visitée). Selon les règles prédéfinies et les paramètres à un instant T (connexion disponible ou non, date de dernière consultation de la page, etc.), un Service Worker permettra d'aller chercher les données d'une page web en mémoire du navigateur ou au niveau des serveurs.

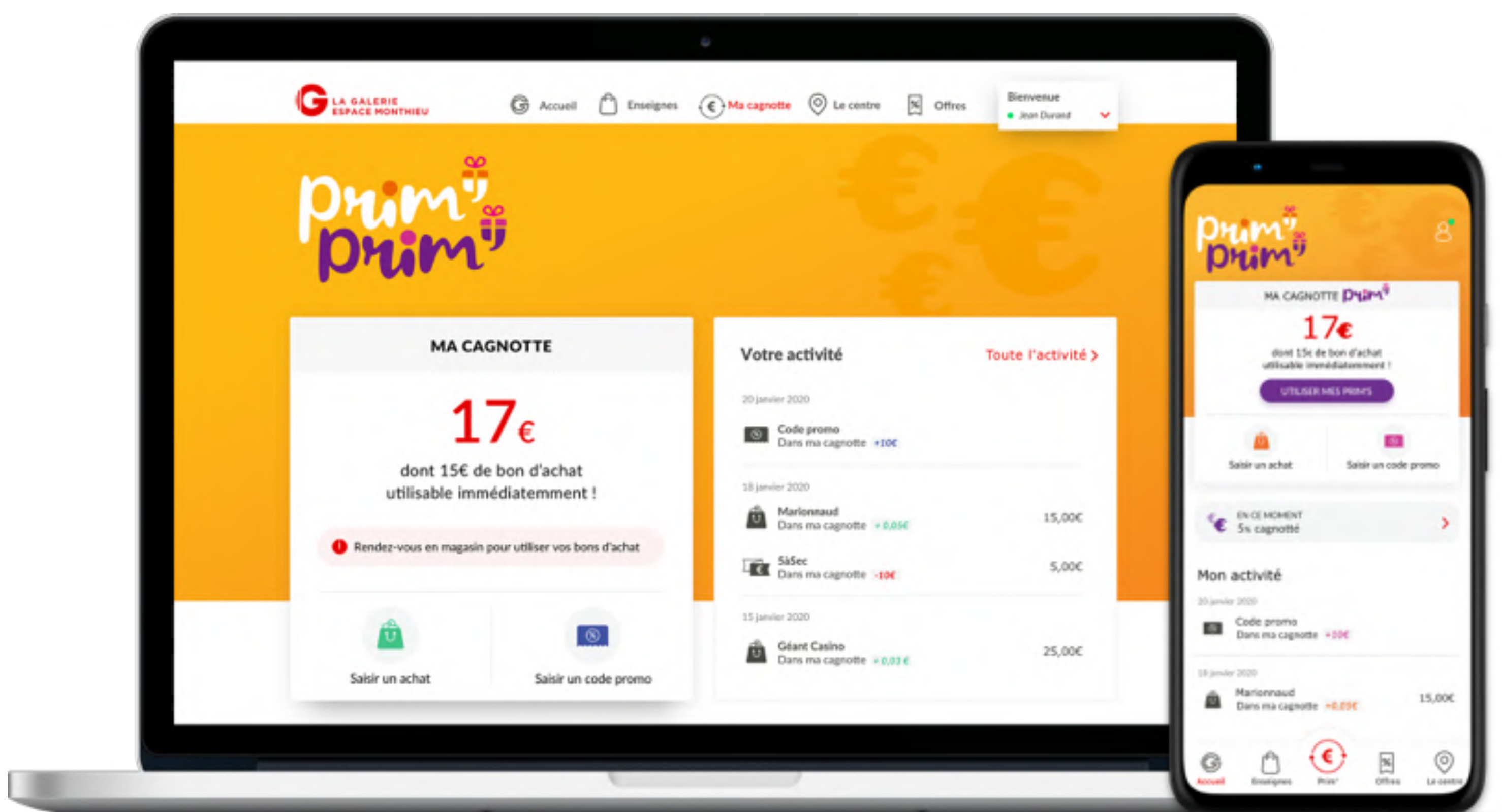
Avec un effort marginal, un site web moderne pourra ainsi être encore plus performant, accessible en partie en mode hors ligne et même disposer des fonctionnalités avancées (22) (push notifications...).

Mais une PWA reste avant tout un site web, et même si un raccourci peut être créé sur l'écran d'accueil de son smartphone, elle ne peut pas être distribuée sur les magasins d'applications. Conscient de cette limite, Google va un cran plus loin depuis 2019 avec la notion de Trusted Web Activities (TWA), renommée récemment en Web in Play.

Avec cette nouvelle technologie, il est possible de publier sur Google Play Store (23) un APK (format des applications Android) créé à partir d'une PWA.

Pour faciliter cette démarche, Google met même à disposition un outil (Bubblewrap (24)) générant automatiquement le package (à noter aussi la possibilité d'utiliser le site PWA Builder initié par Microsoft (25)). À moindres coûts et sans contraintes sur le langage de développement, un site web est ainsi propulsé sur Play Store.

Exemple de TWA réalisée pour Mercialys par USERADGENTS



²² Voir notre précédent article sur les PWA

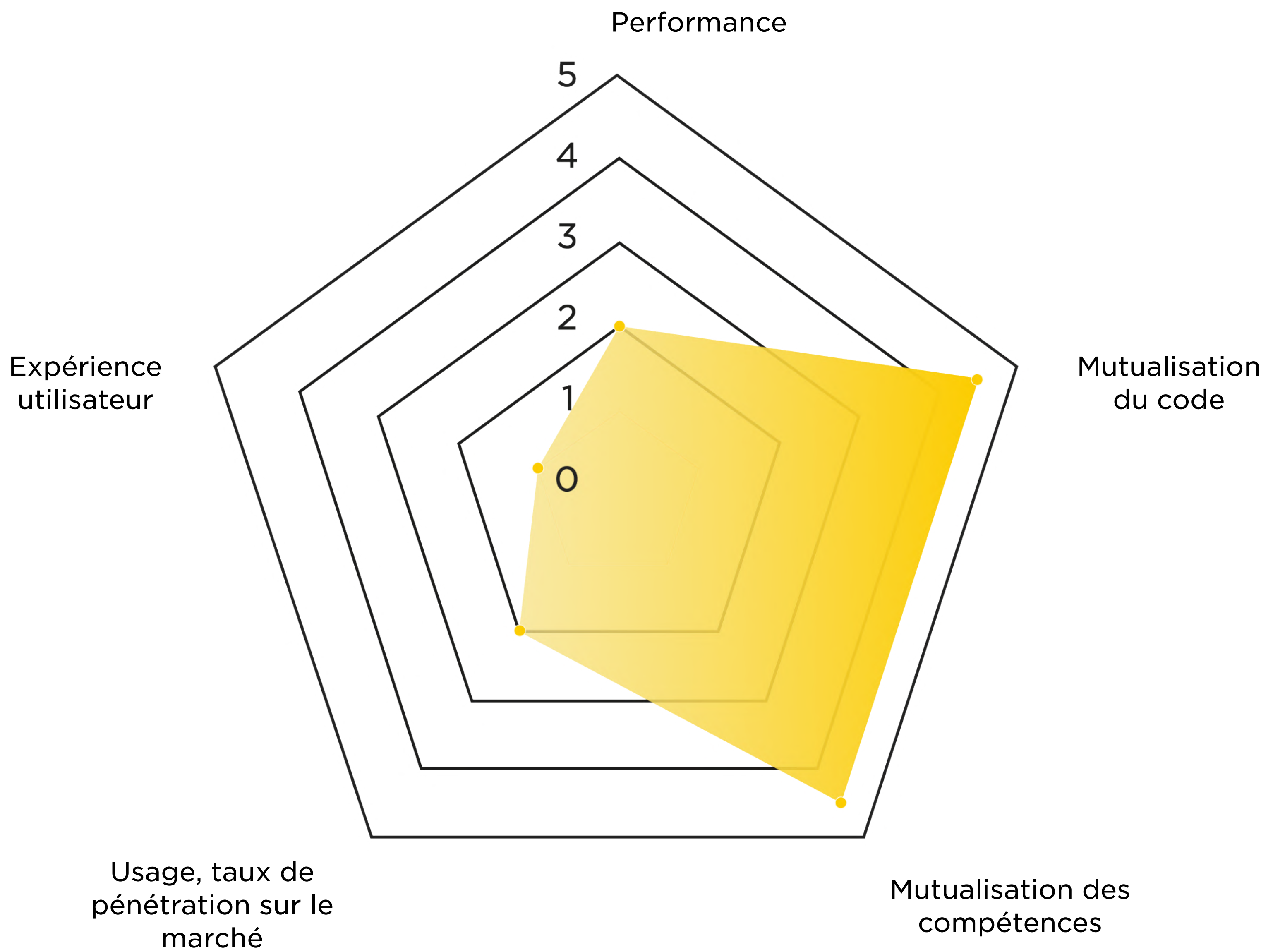
²³ L'application peut également être publiée sur le store

Samsung mais pas sur App Store

²⁴ Bubblewrap

²⁵ Voir <https://www.pwabuilder.com/>

| IONIC



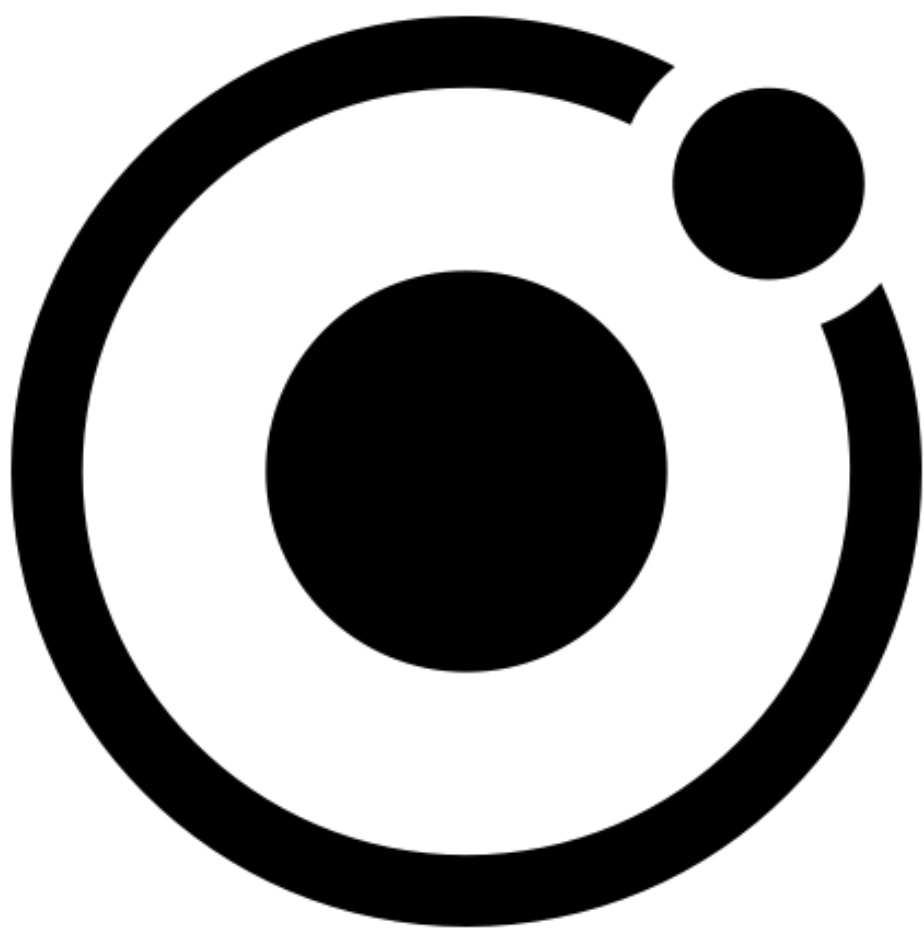
Ionic, le survivant

Dès les années 2009, PhoneGap (devenu plus tard Apache Cordova) propose aux développeurs web de créer des applications mobiles cross-plateformes à partir d'un unique code web.

Comme pour les webviews, les applications créées avec cette technologie s'exécutent dans un container (wrapper) adapté à chaque plateforme mobile.

Mais PhoneGap va plus loin en donnant accès aux capacités matérielles du smartphone (ex. caméra, gyroscope, etc).

Si PhoneGap est depuis tombé aux oubliettes, la technologie permettant de faire le pont avec les capteurs du smartphone (Cordova Bridge) a survécu et est restée jusqu'à très récemment à la base d'une autre plateforme appelée Ionic.



À la différence de PhoneGap, Ionic propose une bibliothèque de composants graphiques (boutons, barre de navigation, etc.) responsive et proche du rendu graphique des composants natifs iOS et Android. Ces composants graphiques sont des composants web et ne sont pas traduits (à la différence de React Native) en composants natifs lors de l'exécution de l'application. Le rendu d'une application Ionic est donc plus proche d'un site web responsive que d'une application native iOS ou Android.

Les évolutions récentes de Ionic lui ont permis de survivre dans cet univers de plus en plus concurrentiel :

- L'ouverture aux framework Vue.js et React ⁽²⁶⁾, Ionic était initialement basé sur le framework javascript Angular.
- Le remplacement du vieillissant Cordova Bridge par Capacitor pour l'accès aux composants matériels du smartphone ⁽²⁷⁾.
- La possibilité depuis fin 2021 avec Ionic Windows ⁽²⁸⁾ de générer des applications Windows pour les versions Windows 10 et +.

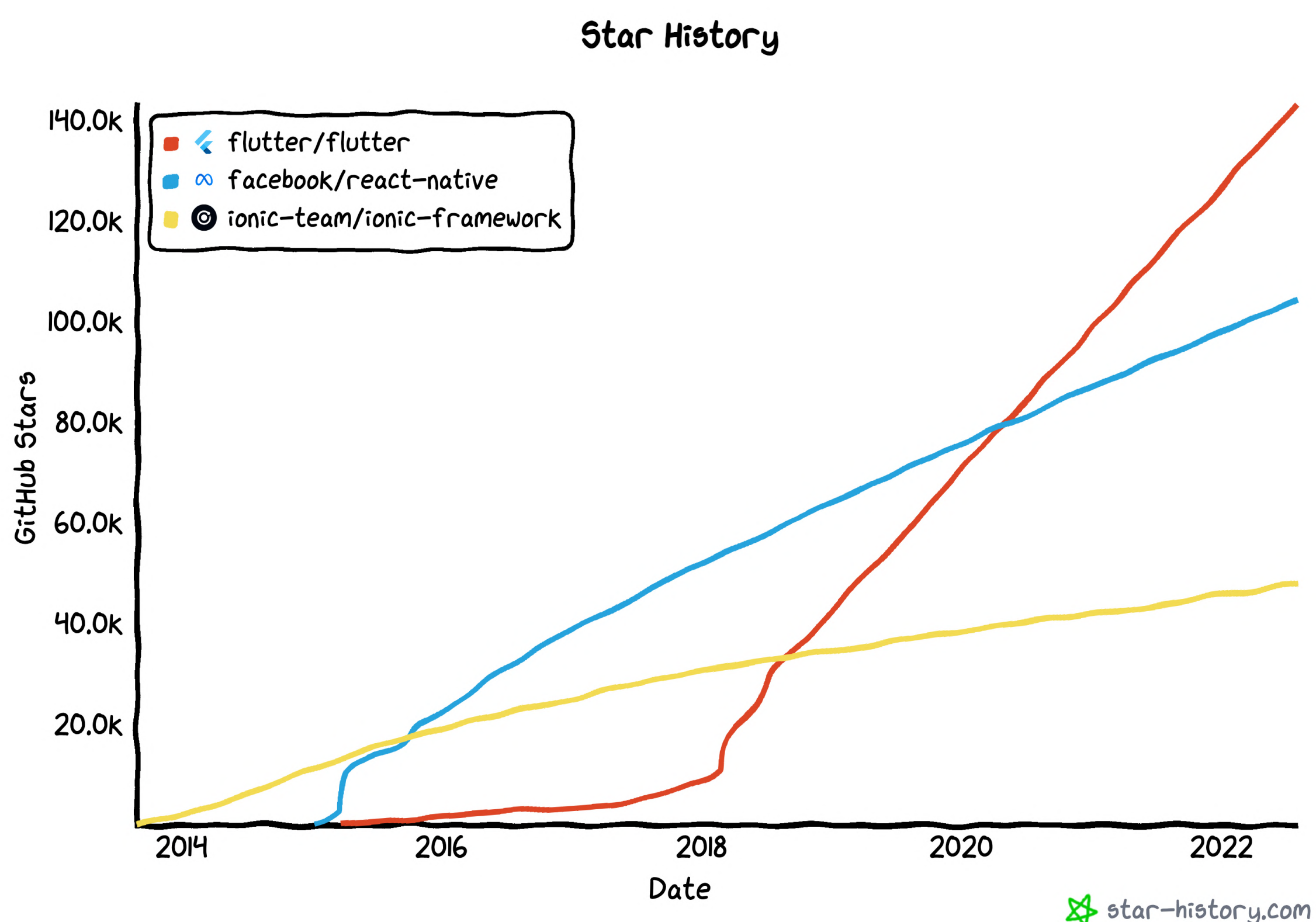
²⁶ Voir les publications de 2019 sur le blog Ionic : *Announcing Ionic React* et *Announcing Vue Ionic Beta*

²⁷ Pour plus de détails techniques, se référer au site de présentation de Capacitor

²⁸ Ionic's official, fully supported integration for deploying native Windows desktop apps

Si, de par sa construction, les performances de Ionic resteront toujours en deçà de celles des applications hybrides ou natives, Ionic permet d'aller un cran plus loin dans la mutualisation en adressant l'ensemble des plateformes mobiles et web.

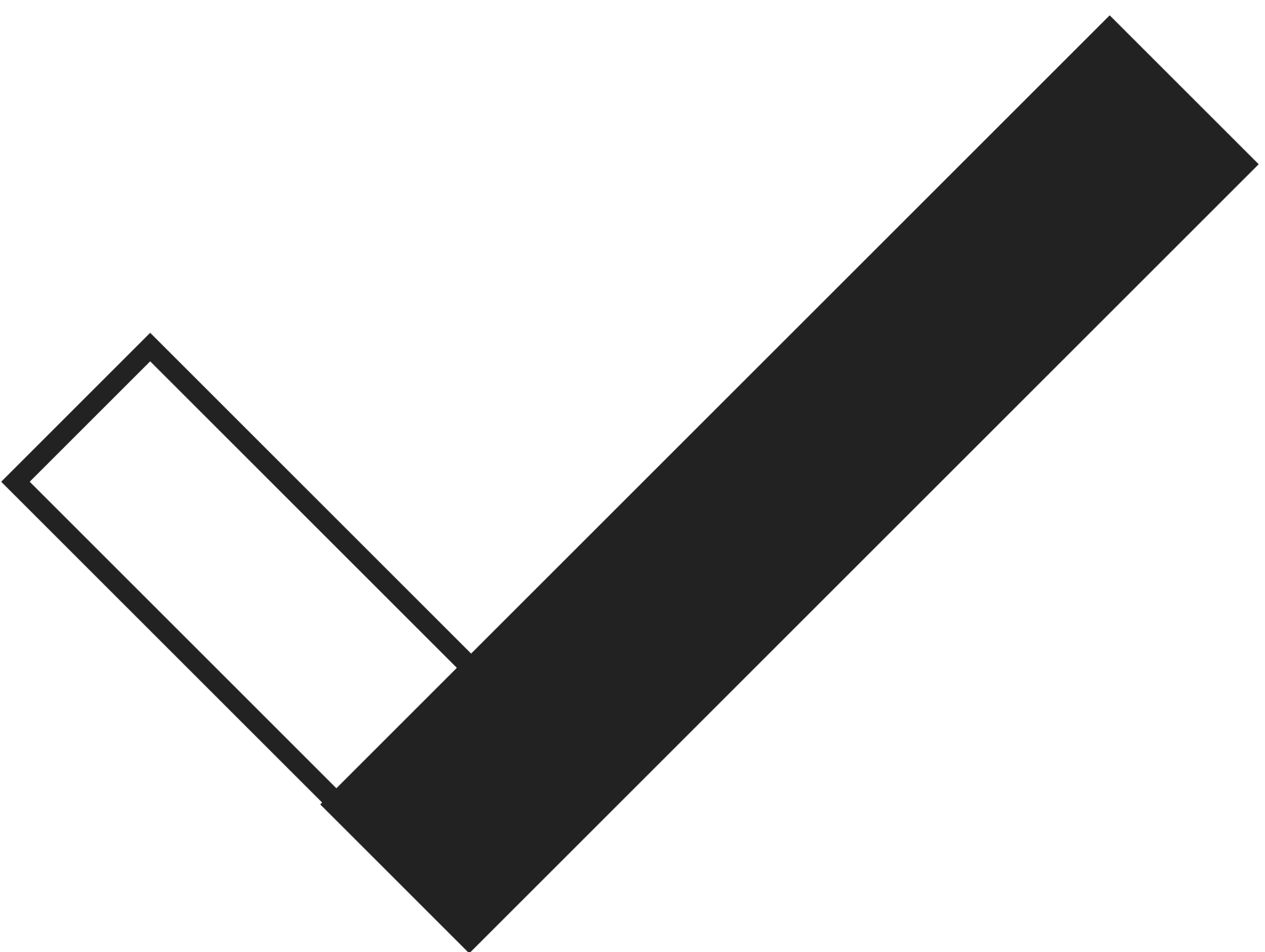
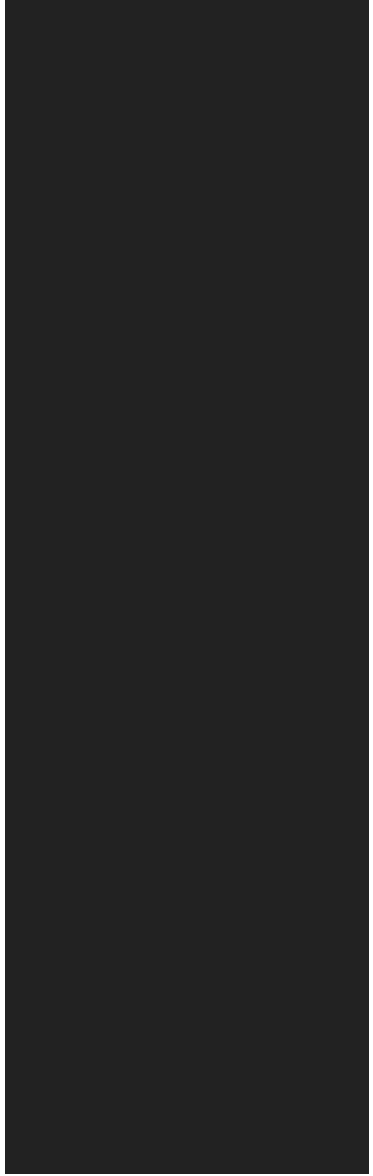
Toutefois, la popularité de Ionic reste faible en comparaison de celle de React Native et de Flutter. (29)



²⁹ GitHub star history Ioni, Flutter, React Native

is

curious



COMMENT FAIRE LE BON CHOIX ?

Déjà long, le panorama dressé jusqu'ici reste incomplet.

Nous aurions pu ajouter **.Net Maui** à la liste (solution de Microsoft remplaçant de **Xamarin**), **Scade** (l'équivalent très peu populaire de Kotlin Multiplatform côté Swift) ou encore le **framework Weex** (initié par la société chinoise Alibaba).

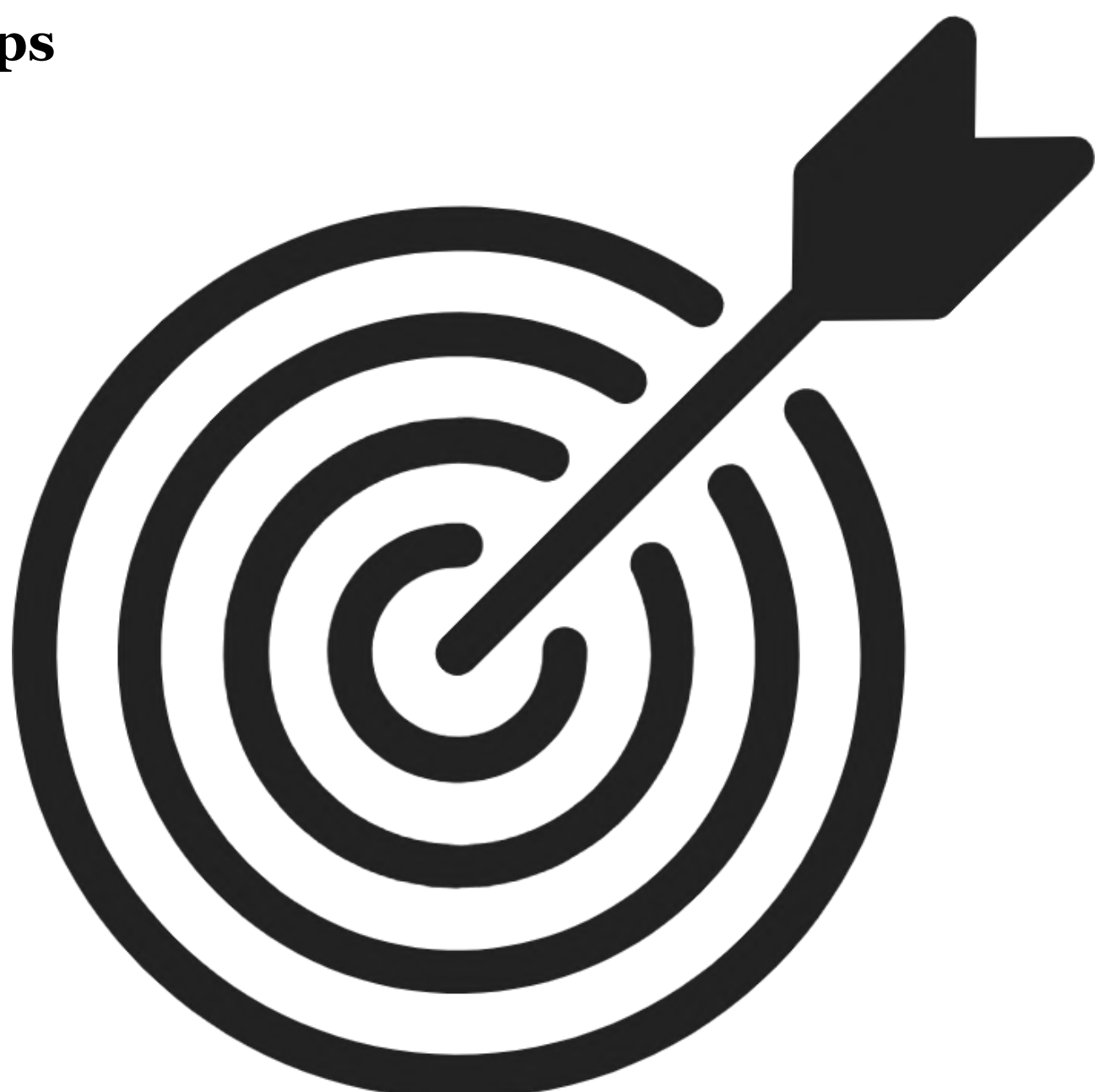
Le nombre de solutions qui cohabitent atteste de la viabilité des différents modèles selon les contextes propres à chaque projet.

Lors de la définition de votre stratégie mobile, s'il est important de dresser une photo à date du marché, il est aussi fondamental de considérer votre contexte :

- + Quelles sont les **compétences** dont vous disposez ?
- + Quel est le **niveau de mutualisation** souhaité avec le web ?
- + Quelles sont les **fonctionnalités** attendues ?
- + Quelle est l'importance accordée à l'**expérience utilisateur** (fluidité de navigation, parcours hors ligne...) ?
- + De quel **budget** disposez-vous ?



À lire aussi sur Emarketing.fr :
notre article « Comment bien choisir la techno de ses apps mobiles en 2020 ? »



Pour vous aider dans votre choix, le tableau ci-dessous propose une évaluation des différentes possibilités. Même si elle est discutable, cette grille permet d’évaluer les forces en présence.

	Native	Cross-platform natif			Hybride		Web embarqué	
Technologie et langage	Native iOS (Swift), native Android (Kotlin)	Kotlin Multiplatform Mobile (Kotlin)	Flutter (Dart)	Xamarin (.Net, C#)	Nativescript (Javascript / Typescript, XML)	React Native (Javascript, JSX)	Ionic	TWA, Webview (HTML, CSS, Javascript, Typescript)
Type d'application généré	Compilée	Compilée			Interprêtée via Javascript Virtual Machine JavaScriptCore, Google V8		Interprêtée via navigateur	
Partage de code	0%	30% à 60%	70% à 90%		70% à 90%		100%	
Composants graphiques, interfaces	Native iOS et Android	Native iOS et Android	Flutter widgets	Xamarin Forms, UI (XAML)	Nativescript UI component (XML), CSS	React Native UI Components	Web (HTML, CSS)	
Exemples	Applications fournies avec l'OS	VMWare, Quizlet	The New York Time (puzzle gaming app), Tencent	Alaska Airlines, FreshDirect, UPS		Facebook, Instagram	Airbus (B2B), Sworkit	L'Equipe

TAKE AWAY

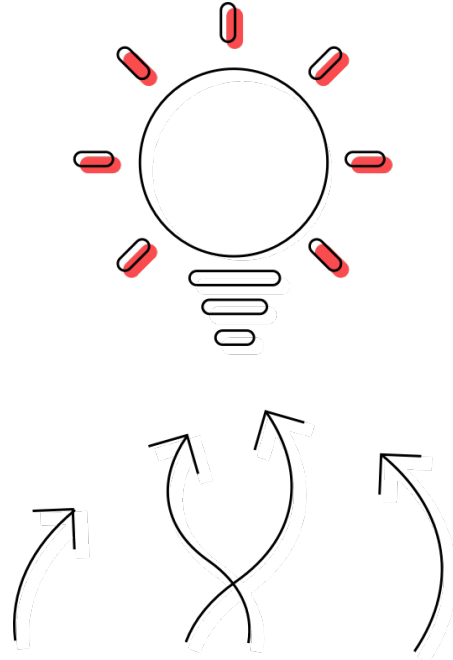
Les évolutions sur le monde du mobile seront encore nombreuses dans les prochaines années. Personne ne sait prédire quelles technologies deviendront incontournables demain. **Kotlin Multiplatform** est prometteur mais encore confidentiel. L'engouement pour **Flutter** est certain mais pourrait retomber. **React**, le grand frère de React Native, est challengé sur ses terres par Vue.js. Même si les géants américains (Google en tête) et chinois battent la mesure, les développeurs restent sur ce domaine faiseurs de rois.

On peut cependant tirer quelques conclusions du marché actuel :

- La **performance brute** n'est plus le facteur différenciant majoritaire. Les performances des solutions cross-plateformes d'aujourd'hui n'ont plus rien à voir avec celles des années 2010 et continuent de s'améliorer.
- Les **approches mixtes** sont toujours possibles, souvent souhaitables (applications natives encapsulant des webviews, code React Native complété par des développements natifs, Kotlin Multiplatform Mobile couplé à Swift pour iOS).
- **Flutter** semble l'emporter face à **React Native** dans la bataille du cross-plateforme 'premium' mais React Native bénéficie encore d'une base installée bien plus importante et de plus de recul.
- La mutualisation complète entre le web et le mobile sera atteinte à moyen terme. **Ionic** et **Flutter** le permettent déjà mais est-ce que le marché est prêt pour cette transformation ? Leur usage sur le web reste encore très marginal sur le B2C.

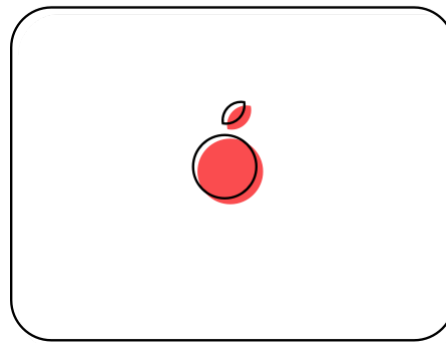
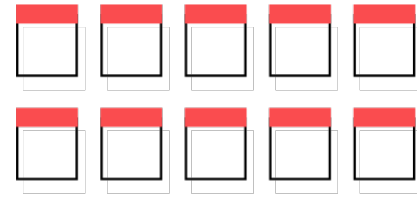
ACCOMPAGNEMENT

Design



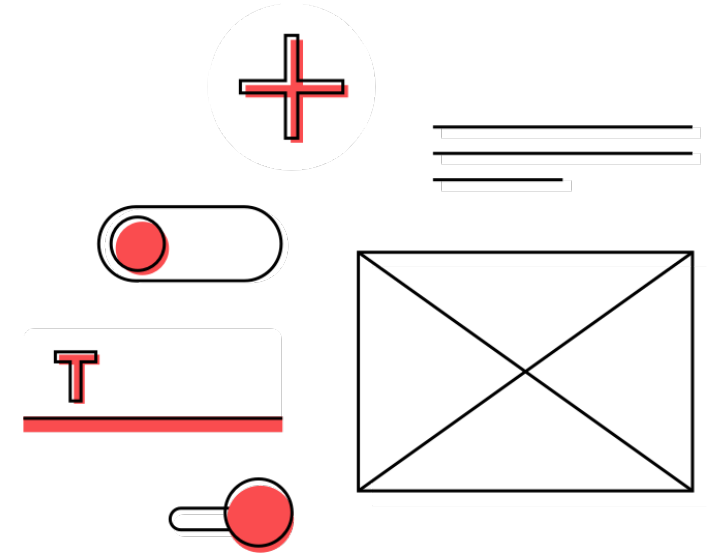
Stratégie

Comprendre les enjeux
& définir la vision du service



UX Design

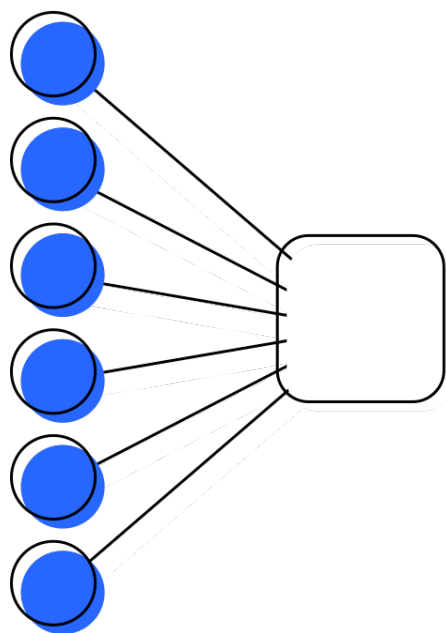
Identifier les besoins
& concevoir les expériences



UI Design

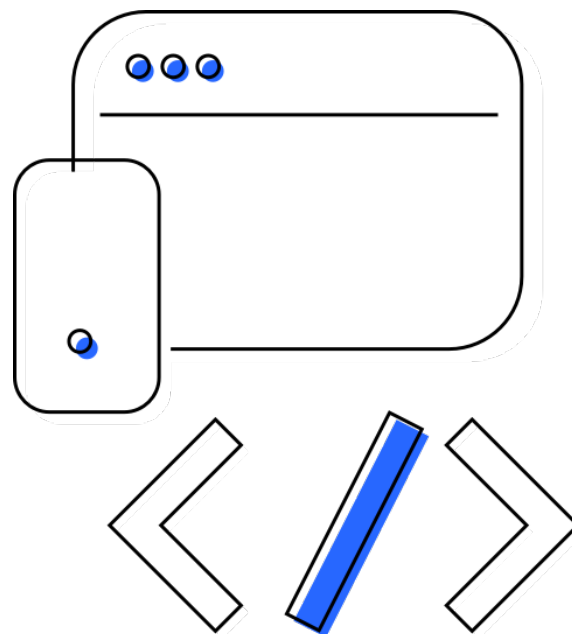
Designer les interfaces
& incarner la marque

Tech



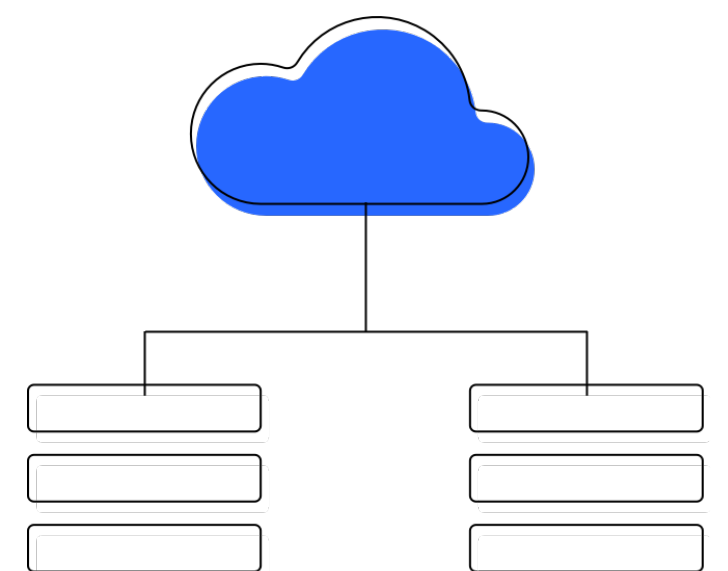
Architecture & solutions

Définir l'architecture, identifier
les briques technologiques



Développement front (web & app)

Développer avec les langages
les plus performants



Back-end & plateformes

S'appuyer sur des solutions
modulaires & découplées



USERADGENTS

Agence digitale mobile first

8, rue de la Rochefoucauld
75009 Paris

Pour nous contacter :



+33 1 77 75 65 90



uainfo@useradgents.com



www.useradgents.com



in



f