TheCodingMachine
TCM://

Développez vos applications mobiles en React Native

—

Du choix technologique en passant par le développement,

Jusqu'au déploiement !

Hugo Averty, Directeur Régional Lyon
Aurélien Mutin, Directeur Technique Lyon

Mars 2019

Introduction

Comme tout ce qui semble trop évident, il est pertinent de s'y attarder un instant… car une application mobile n'est pas seulement une application pour smartphones !

Déployer aujourd'hui une plateforme web demande de supporter les principaux navigateurs dans leurs principales versions ; déployer une application mobile implique d'autres enjeux.

En effet, une App se doit d'être portée sur :

- Différents systèmes comme Android, iOS ou Windows, qui comme nous le verrons plus loin, possèdent chacun en langage de développement spécifique,
- Différentes versions de systèmes comme Android 7, Android 8, iOS11, iOS12…
- Différents devices (téléphones, tablettes, télévisions) et même différents modèles : iPhone X et son encoche, Samsung Galaxy A8s et son trou dans l'écran… les écrans de demain seront même pliables !

Il est donc important dans un premier temps de comprendre ce marché volatil et très à même aux changements. Vous pouvez parfaitement déployer une App uniquement pour smartphones ou uniquement pour tablettes par exemple, mais il est plus commun de vouloir englober la majorité des devices.

Ce livre blanc vous permettra de mieux appréhender les grands changements de l'écosystème mobile qui s'opèrent. Il vous aiguillera dans vos choix et à comprendre les enjeux technologiques pour mener à bien le développement et le déploiement de vos applications mobiles.

1 Fonctionnement général d'une application mobile

Développer une application mobile demande d'effectuer des choix technologiques qui peuvent faire perdre le fil aux plus assidus. Hybride, native, web-App, sont des termes que vous entendrez dès que vous mettrez le premier pas dans le monde des applications mobiles. Attardons-nous d'abord sur l'architecture globale d'une application, qui sera commune, qu'importe les choix technologiques.

Une application mobile est ce que l'on dénomme un « **front** » ou « front-end ». Il s'agit de **la partie visible de l'iceberg**, qui est entre les mains de l'utilisateur final, ce qu'il verra et utilisera au quotidien. L'intelligence d'une application se résume donc à une intelligence fonctionnelle et ergonomique.

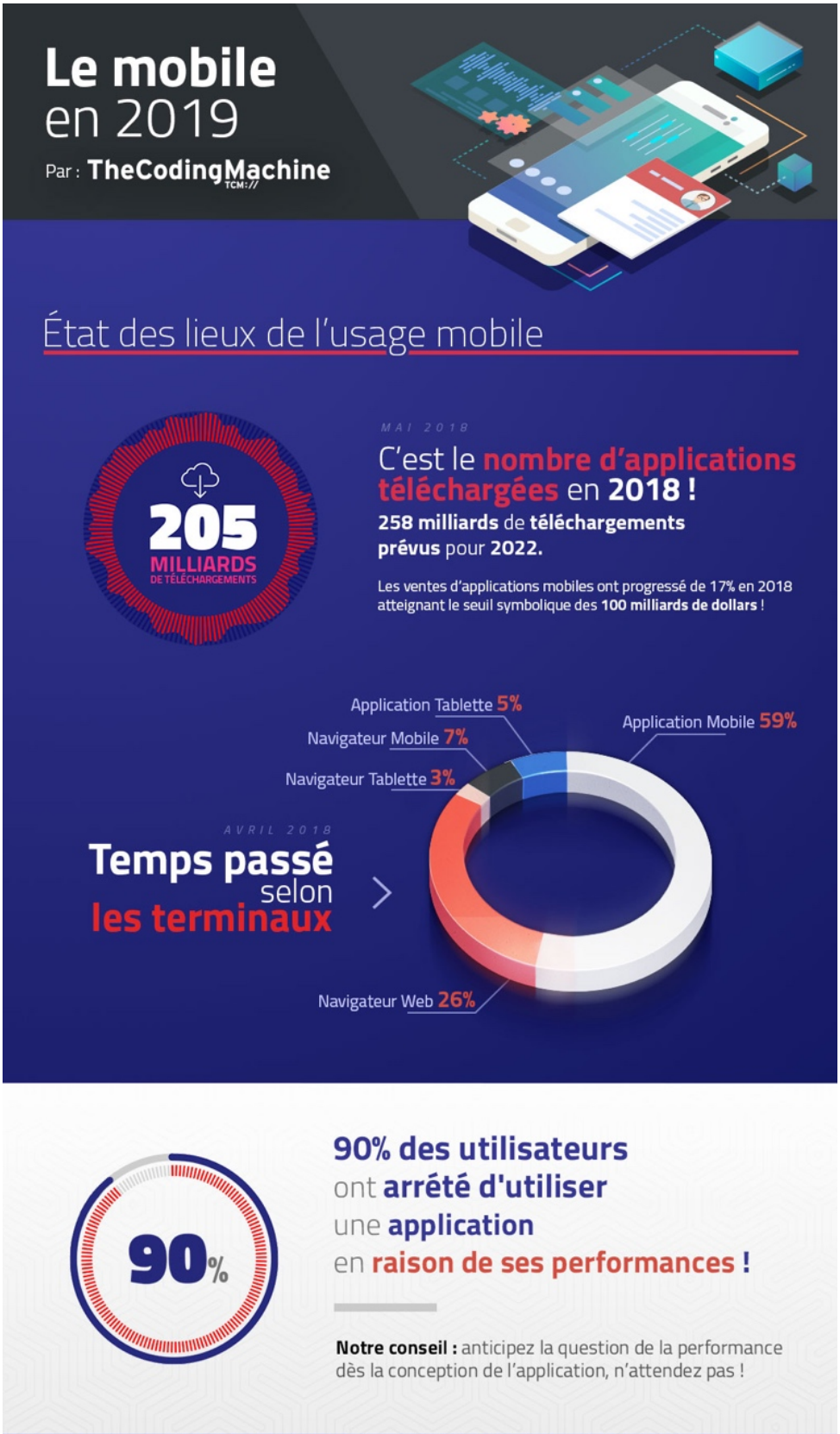
Toute l'intelligence métier se retrouve dans une partie « **back** » ou « back-end », isolée du « front ».

<u>Prenons un exemple très concret :</u>

> Vous venez de faire l'acquisition d'un magnifique moteur de voiture. Félicitations !
> Ce moteur possède tout un tas de mécanismes très complexes et très techniques. Les experts vous en parleront dans leur jargon bien à eux : cylindre, vilebrequin, piston, etc. Le moteur est le « back-end ». Il est intelligent et technique, un peu magique, le principal étant qu'il soit capable de faire avancer une voiture.
>
> Cependant votre moteur seul possède peu d'intérêt. Vous décidez donc de le monter sur une jolie tondeuse (une de celles où l'on peut s'asseoir dessus). Elle est verte, possède quatre roues et un siège en cuir. C'est votre « front-end ». Sans le moteur, elle n'avancera pas toute seule, mais elle reste agréable à l'œil malgré tout, et le siège est très confortable. Ergonomique et design caractérisent ce « front ».
>
> Cependant, votre moteur semble un peu surdimensionné pour votre tondeuse, qui avance, mais un peu trop vite. Vous décidez alors d'installer votre nouvelle acquisition sur une voiture. Une voiture de bonne qualité, bleue, avec quatre roues, deux sièges à l'avant et une banquette arrière des plus agréables. Encore un nouveau « front-end ».

Comme vous pouvez le voir, nous avons deux « fronts » qui peuvent se connecter avec le même « back ». Cette notion est très importante dans l'univers mobile, car il est fréquent d'avoir une seule plateforme « back-end » sur laquelle vont se connecter à la fois une application mobile et un site web, deux « front-end » distincts.

Le mobile en quelques chiffres




Comparatif des technologies & frameworks
Comparatif sur 4 composantes : les meilleurs peformances, la solution la plus intéressante en terme de prix, les possibiltés offertes et la simplicité de compréhension.
PERFORMANCES
Basé sur le temps d'affichage, et les images par seconde (fps)
OPTIMISATION COÛTS & DÉLAIS
Coûts de développement, de maintenance et du nombre de plateformes.
POSSIBILITÉS
Possibilité technique de l'application : VR, jeux, animations, sur mesure, etc.
SIMPLICITÉ POUR LES DÉVELOPPEURS WEB
Facilité pour les développeurs web ne connaissant le développement mobile.
Natif
iOS : Swift / Objective-C
Android : Java
React-Native
Techno cross-plateform développé par Facebook
Flutter
Techno cross-plateform développé par Google
Hybride
Cross-plateform : Cordova, Xamarin ,etc.
Pratiques d'utilisation
À chaque besoin il existe une ou plusieurs technos appropriées.
Natif
Application optimale en termes de rendu et performance. Convient particulièrement aux applications B2C. Technologie éprouvée, nécessite des compétences (développeurs) dans plusieurs langages (Swift & Java). Budget important.
React-Native
Application performante avec un rendu très proche du natif. Convient aux applications B2C & B2B, quel que soit le budget. Technologie développée par Facebook en 2015 et en pleine croissance.
Flutter
Application performante avec un rendu très proche du natif. Convient aux applications B2C & B2B, quel que soit le budget. Lancée après React-Native (2 ans après) et poussée par Google, elle ne bénéficie pas encore de la communauté des trois autres (natif, hybride, RN).
Hybride
Application type POC, événementielle ou jetable, pouvant être plus facilement portable sur le web. Convient particulièrement aux applications B2B à faible budget. Technologie en perte de vitesse avec l'arrivée des frameworks cross-plateform (React-Native, Flutter, etc.)
TOP PERFORMANCE
MEILLEURES PERFORMANCES
Natif
NUMBER #1 ONE
MEILLEURE RENDU / COÛT
React-Native
MONEY MAKER
MEILLEURE PETIT BUDGET
Hybride
Qui utilise quoi ?
Natif
Airbnb anciennement React-Native
LinkedIn anciennement Hybride
BlaBlaCar
Slack
Spotify
Candy Crush
Hybride
Picturex
JustGiving
Evolve
Alaska Airlines
Olo
Storyo
React-Native
Facebook anciennement Natif
Instagram
Tesla
Pinterest
Skype
Uber
Flutter
Google Ads
Reflectly
Alibaba
AppTree
JD Finance
Hamilton
Publié en 2019
Sources : Com Score 2017, Com Score 2018, App Store

2 Natif vs. Hybride, le combat des géants

Les applications natives

Développer une application native signifie que vous êtes en train de développer **une application propre à un environnement et une plateforme donnée** (en utilisant le langage préconisé par le constructeur). Les applications natives sont arrivées avec les premiers smartphones et sont à l'origine de l'avènement de l'iPhone.

Par exemple :

- sur iPhone nous utiliserons le langage : Objective-C ou plus récemment Swift
- sur Android nous utiliserons le langage : Java

Les applications mobiles natives ont l'avantage d'offrir aux utilisateurs des **performances et un rendu optimal :** elles sont de manières générales plus rapides et fluides, améliorant ainsi l'UX - expérience utilisateur. Aussi, les limitations sur les applications natives sont nulles comparées à celles proposées sur de l'hybride.

Cependant, **elles nécessitent un langage de développement pour chaque plateforme** (Android, iOS). Ainsi, le temps **de développement et le coût sont donc proportionnellement plus importants** et se retrouvent quasiment doublés. Aussi, elles requièrent des développeurs sur plusieurs langages si vous décidez de faire du multiplateforme (Java, Swift).

Les applications hybrides

Ces applications sont développées dans des langages initialement utilisés - et prévus - dans le web : HTML, CSS, JavaScript. Vulgairement, **elles encapsulent une page web**, mais aussi elles permettent **d'intégrer l'ensemble des fonctionnalités du téléphone** moyennant des plug-ins (notifications, géolocalisation, etc.)

Les applications hybrides ont l'avantage d'être plus facilement portables sur le web dans la mesure où elles sont déjà développées dans des langages web. Le temps de formation sur ces technologies est donc faible pour les développeurs qui sont habitués à ces langages.

Par exemple il est possible d'utiliser Apache Cordova, un framework de développement mobile open-source. Il permet d'exploiter les technologies Web courantes telles que HTML5, CSS3 et JavaScript

Comment choisir ?

Soyons clairs, il n'y a pas vraiment de combat entre le natif et l'hybride : ils correspondent tous les deux à des besoins spécifiques !

Le natif a un coût de développement et de maintenance plus élevé si vous avez prévu de déployer l'application sur plusieurs terminaux (Android/Apple). Trivialement, son coût pourrait être quasi doublé. En revanche son rendu optimal en termes de performance, permettra de proposer la meilleure expérience possible.

À l'inverse, l'Hybride a un coût de développement et de maintenance réduit, d'autant plus si vous avez prévu de porter votre application sur Android, Apple et sur le web. En revanche, l'hybride encapsule une page web dans une application et ne propose pas la liberté que peut proposer le natif. Le plus gros reproche fait aux technologies hybrides concerne ses performances qui ne permettent pas d'avoir une expérience utilisateur fluide et confortable.

L'alternative née de Facebook

Depuis maintenant quelques années est apparu un nouveau type de framework. Il reprend le principe de l'hybride : développer une application Android, Apple ou Smart TV avec un seul et même langage.

L'objectif de cette nouvelle technologie est de garder un bon niveau de performance en proposant un langage Cross-Platform.

Les technos cross-platform :

Framework créé par Facebook à la suite d'un Hackathon en 2015., React Native est basé sur React, une librairie JavaScript développée deux ans auparavant par un ingénieur Facebook (Jordan Walke).

Flutter, le framework de Google permettant de concevoir des applications multiplateforme pour Android et iOS, est désormais disponible en Release Preview 2.

Conçu et développé par Progress, les applications NativeScript sont créées à l'aide de JavaScript ou de tout langage qui transpile du JavaScript, tel que TypeScript. NativeScript prend en charge les frameworks JavaScript, Angular ou Vue.js.

3 Développez avec React Native

Comme Facebook le dit très bien :

> ‹ Avec React Native, vous ne construisez pas une Web-App mobile ni une App HTML ou encore une application hybride.
>
> Vous construisez une application mobile qui n'est pas différenciable d'une application utilisant de l'Objectif-C ou du Java. React Native utilise les mêmes fondamentaux de construction et blocs d'UI - *User Interface* - comme c'est le cas sur les applications iOS et Android.
>
> En bref, React Native est un langage unique de développement permettant de générer des applications identiques à des applications natives. ›

React Native is a native **cross-platform solution**,
learn once, write anywhere.

La naissance de React Native

Du point de vue d'un développeur, on peut dire que les origines de React Native sont assez récentes puisqu'elles remontent à l'été 2013, et comme beaucoup de projets elle est issue d'un side-project : React.js. React Native est donc la descendance de React.js, tous deux enfants de Facebook.

La première version de React Native est née pendant un hackaton, cependant la première communication publique n'est quant à elle apparue qu'en janvier 2015 à la React.Js Conf. C'est lors de la F8 conférence, en mai 2015, que le lancement de React Native est annoncé officiellement et que le projet Open-Source est lancé sur GitHub.

Cette première version ne supportait qu'iOS mais la version Android viendra rapidement après en septembre 2015.

Comment ça marche ?

Lorsqu'une application React Native est lancée, il existe deux threads qui tournent et qu'il faut retenir :

- L'un d'eux est le "main thread", qui s'exécute également dans les applications développées en langage natif. Il gère l'affichage de l'interface utilisateur et traite les interactions utilisateur (Swipe, Pinchzoom, etc.).

- L'autre est spécifique à React Native. Sa principale tâche est d'exécuter le JavaScript, il traitera l'ensemble de la mécanique métier de votre application. Il définit également la structure et les fonctionnalités de l'interface utilisateur.

Ces deux threads ne communiquent jamais directement et ne se bloquent jamais. Entre eux se trouve un bridge qui va leur permettre de communiquer : c'est en quelque sorte le cœur de React Native.

Ce pont a trois caractéristiques importantes :

- Asynchrone : Il permet une communication asynchrone entre les threads. Cela garantit qu'ils ne se bloquent jamais.
- En lot : Il transfère les messages d'un thread à l'autre de manière optimisée.
- Sérialisable : Les deux threads ne partagent ni ne fonctionnent jamais avec les mêmes données. Au lieu de cela, ils échangent des messages sérialisés.

> Si vous voulez plus de détails sur l'architecture de React Native, nous vous invitons à aller voir la présentation de Tadeu Zagallo, Software Engineer à chez Facebook & React-Native
> https://www.youtube.com/watch?v=Ah2qNbI4OvE

Et Windows dans tout ça ?

React Native ne permet pas à ce jour de développer nativement des applications Windows.
Il est important de noter que les parts de marché de Windows sur le mobile sont extrêmement faibles. En novembre 2018, Android et iOS se partagent 98,63% du marché français (96,79% à travers le monde), contre seulement 0,54% pour Windows (0,39% à travers le monde).[1]

[1] *Source : http://gs.statcounter.com/os-market-share/mobile/france/#monthly-201812-201812-bar*

N'importe quelle application grand public trouvera donc sa cible auprès des utilisateurs Android et iOS. Il n'est donc pas nécessaire à ce jour de déployer sur Windows compte tenue de l'effort trop important et des retombées trop faibles.

Cependant, il peut s'avérer intéressant de déployer des applications Windows pour des besoins très spécifiques : potentiellement pour des applications internes et des entreprises possédant un parc de téléphones Windows.

Du fait de la popularité de React Native, Microsoft a mis en place un plug-in officiel (https://github.com/Microsoft/react-native-windows) permettant de déployer des applications React Native pour Windows 10, Windows 10 Mobile et Xbox One (UWP).

Est-il possible de déployer mon appli React Native sur le web ?

Non ! Et voici pourquoi :

Admettons tout d'abord un fait et oublions tous les aspects techniques : l'ergonomie d'une application mobile est absolument spécifique. La navigation, l'UX et l'UI que nous retrouvons sur une application mobile sont donc rarement adéquats sur un site web.

Également, une application a souvent un but très spécifique, fortement lié au device, comme prendre des photos, scanner des documents, géolocaliser l'utilisateur, … qui sont des fonctionnalités propres à une utilisation mobile. De plus, si votre application n'a que pour seul but d'être un site web qui tient dans la poche, il est fort probable qu'un site « responsive » correctement pensé et structuré soit suffisant.

Cependant, il reste absolument fréquent et logique de vouloir porter son application mobile sur un équivalent web. Pas de panique côté budget ! Il ne faut pas oublier que toutes les logiques métiers sont isolées – en back-end – et doivent être réutilisées par le site web, qui ne sera qu'un nouveau front. La plupart du temps, ce site web ne sera pas identique en terme d'ergonomie et pourra proposer des fonctionnalités différentes si nécessaire.

Vous trouverez probablement après quelques recherches qu'il existe des plug-ins pour transformer une application React Native en application web, comme par exemple : https://github.com/necolas/react-native-web

Avant de vous lancer tête baissée, gardez à l'esprit les paragraphes précédents : sachez avant tout que la majorité des fonctionnalités de votre application ne fonctionnera pas avec ce genre de plug-ins (dès lors que vous utilisez un plug-in un peu spécifique), et nous restons persuadés que vous ne voulez pas d'une application mobile comme site web.

Comment déployer mon appli sur les stores ?

Cette question fera très certainement l'objet d'un livre blanc à part entière, mais il semblait très important de l'aborder ici.

Côté sécurité, une appli mobile est-elle fiable ?

Si vos partenaires techniques sont fiables, alors votre application le sera également ! Car certaines notions techniques sont indispensables à la sécurité d'une application. Il est donc primordial que votre partenaire ait connaissance du minimum de prérequis essentiels à la sécurisation de votre application.

En voici quelques exemples pour illustrer ce propos.

Nativement, il est sage de partir du principe que n'importe quel utilisateur est capable de lire le code d'une application. Il est donc **interdit de stocker des données sensibles** au sein d'une App. Pas de clés privées pour se connecter à des serveurs ou des bases de données, pas de mot de passe utilisateur sauvegardé pour conserver l'authentification, etc. Une application doit être des plus « naïves » possible et ne rien connaître des données critiques.

Une application pourra accéder à beaucoup de données du téléphone (photos, géolocalisation, etc.). Le minimum vital en termes de chiffrement des échanges est donc requis si l'application fait transiter ces données vers un back-end.

Un dernier exemple : si votre application gère des données très sensibles (application bancaire, réseau social, documents personnels, etc.), déconnecter l'utilisateur de sa session dès que possible est judicieux. Votre utilisateur aura pour réflexe de fermer l'application sans se déconnecter manuellement en appuyant sur un bouton spécifique, ou éteindra tout simplement son téléphone. L'application doit donc déconnecter l'utilisateur elle-même dès que ces actions sont effectuées : fermeture de l'App, mise en arrière-plan, ...

4 React Native, côté technique

L'objectif de cette partie est de comprendre comment React Native fonctionne et comment l'utiliser.

Attention, avec un environnement Linux/Windows vous allez pouvoir compiler et émuler votre application uniquement pour Android. En revanche si vous êtes sur Mac, vous aurez la possibilité de tester les deux applications iOS et Android que ce soit via la machine virtuelle ou depuis un device.

Le langage React Native

React Native s'écrit en JavaScript, les développeurs web pourront donc assez rapidement monter en compétence sur ce langage et écrire des applications mobiles natives.

Le fait d'avoir une application écrite en React Native est d'autant plus intéressant si vous disposez de développeurs web internes qui pourront travailler sur votre back-office, site(s) et application mobile. Il utilise le moteur JavaScript Core, il est donc compatible avec les différentes normes JavaScript (ES5, ES6, ES7)

Environnement de développement

Pour débuter sur React Native, rien de plus simple, il est possible de développer sur Windows, Linux et Mac OS. Classiquement, quel que soit votre OS vous devrez installer sur votre machine : **NodeJS / NPM / Yarn / un IDE (Eclipse / Atom / Subime Text, etc...)**

- **Mac : iOS & Android**

Pour développer sur iOS, vous n'avez pas le choix, il faut un Mac et installer :

- Xcode
- Watchman

Il est également possible de faire une application Android et la tester depuis un Mac en installant Python 2 et le Java SE Development Kit (JDK).

- **Windows/Linux : Android uniquement**

La procédure est la même que pour Mac, il faut installer Python 2 et le Java SE Development Kit (JDK). Il est recommandé par Facebook d'utiliser https://chocolatey.org/ pour installer Python 2 et le JDK.

Le fonctionnement de React Native

- **Les composants**
-

La syntaxe de React Native est très similaire à celle de React.js. Bien qu'il existe quelques variantes qui lui sont propres, le principe de composant reste le même. En effet, il faut savoir que le framework a été développé autour de la logique des composants, notion très importante à garder en tête.

React Native met à disposition des composants que vous pouvez consulter via ce lien. Ce qui rend le principe vraiment intéressant, c'est que vous pouvez développer vos propres composants.

Par exemple, lors du développement d'une application mobile, vous allez possiblement avoir besoin de développer un emplacement où il y a le nom de l'utilisateur connecté et son ancienneté.

Nous allons donc avoir un texte avec le nom et le prénom de l'utilisateur ainsi qu'une petite étoile avec son ancienneté qui sera une image.

Il s'agit là d'un groupement de composants React Native que nous allons développer au sein de l'un de nos composants.

```
class UserProfil extends React.Component {
    render() {
        return (
            <View>
                <Text>{ user.fullName }</Text>
                    <Image
                        source={require('./image_path')}
                    />
            </View>
        )
    }
}
```

- **Les vues avec Flexbox**

Pour commencer, Flexbox n'a rien à voir avec Facebook, cette notion existait déjà avant la naissance du framework, toutefois l'outil a été légèrement adapté pour React Native.

Flexbox est indispensable pour les styles : vous savez qu'il existe une multitude de téléphone avec des tailles d'écrans différentes. Cela peut vite devenir un enfer en adaptation pour chaque device pour les développeurs, mais heureusement Flexbox existe ! Vous avez maintenant la possibilité d'ajouter à vos styles une propriété Flex.

Si nous prenons comme exemple de diviser un écran en deux pour y afficher deux contenus différents. Que va-t-il se passer ? Nous allons définir une taille en pixel ? En pourcentage ? Flex se rapproche des pourcentages sauf que nous allons appliquer la propriété Flex aux éléments (un peu comme Bootstrap).

Dans cet exemple nous allons afficher les « ... » au début de l'écran et les seconds « ... » au milieu de celui-ci.

Et cela pour un iPhone 5 ou un iPhone X. La taille de l'écran n'est clairement pas la même, mais le centrage du contenu va automatiquement s'adapter.

```
    render() {
        return (
            <View style={{ flex: 1 }}>
                <View style={{ flex: 1 }}>
                    <Text>
                        { '...' }
                    </Text>
                </View>
                <View style={{ flex: 1 }}>
                    { '...' }
                </View>
            </View>
        )
    }
```

Lien : https://github.com/vhpoet/react-native-styling-cheat-sheet#flexbox

- **Les props**

Les props sont une notion extrêmement utilisée en React Native, comme en React.js. La grande majorité - pour ne pas dire la totalité – des développeurs ont déjà travaillé avec des props. Une props est une propriété d'un composant.

Par exemple sur un composant <Text> la propriété « style={} » est une props.
Voici la liste des props du composant <Text> : https://facebook.github.io/react-native/docs/text.html#props

Ce qui rend les props très populaires, c'est que le développeur peut créer des props pour les composants qu'il a développés. Lorsqu'un composant est appelé, et lorsqu'une props est passé en paramètre, il est très simple de récupérer l'information à l'intérieur du composant.

Si nous reprenons notre exemple de profil utilisateur avec l'étoile nous allons appeler notre composant comme cela <UserProfil/> en ajoutant une props userId :

```
<UserProfil userId={user.id} />
```

Du côté de la classe du composant, il sera très simple de récupérer cette information à l'aide de cette commande :

```
this.props.userId
```

Notez également que l'information envoyée dans la props par le composant parent ne pourra pas être modifié par le composant enfant qui la reçoit. Faites donc bien attention aux informations que vous allez envoyer.

- **Les states**

Les states sont aussi une notion également très importante en React Native. Pour mettre des informations dans le state, nous allons utiliser setState(). Notez qu'à chaque fois que le state est modifié, notre application va s'afficher (render), c'est-à-dire se rafraîchir sans pour autant changer de page.

Toutes les variables du state sont accessibles via cette commande :

```
this.state.[variable]
```

Les states sont très intéressants pour rendre l'application fluide et faire en sorte que les éléments changent en fonction de la navigation et l'interaction de l'utilisateur.

- **La navigation**

En React Native, une question se pose, comment naviguer entre toutes ses pages ? Nous allons parler ici de ReactNavigation.

Pour préparer la base de la navigation de notre application, nous allons créer un composant Navigation qui va lister toutes les vues. Depuis notre App.js, qui est le point d'entrée de l'application, il ne nous reste qu'à appeler notre composant Navigation.

Il sera alors possible depuis notre page principale d'appeler une vue en fonction de notre besoin. Par exemple lorsqu'un utilisateur soumet un formulaire nous pouvons appeler une nouvelle vue pour afficher un récapitulatif.

Il faut noter également que ReactNavigation nous mâche le travail, toutes les vues définies dans le composant Navigation vont être passées automatiquement dans les props des composants enfants. Donc si nous passons de la homepage (avec une liste d'utilisateurs) à la page de détails d'un utilisateur que nous avons appelé UserProfil nous allons pouvoir appeler la vue UserProfil comme cela depuis la vue homepage :

```
this.props.navigation.navigate('UserProfil')
```

- **Redux**

Cette partie est la plus complexe et concerne les stores en React Native. Nous allons vous présenter encore une fois très rapidement ce concept, car cela nécessite de détailler le sujet pour bien comprendre le principe.

Redux va nous permettre d'utiliser les stores, je ne vais pas rentrer dans le détail du code, mais vous devez comprendre qu'après avoir vu les states, on est relativement limité. Un state est utilisé dans une classe et va potentiellement être transmis à un ou plusieurs composants.

Mais si nous voulons un state global qui pourra être utilisé de manière plus générale sans faire passer une variable du state de vues en vue et de composants en composant ? Et bien il existe bien un state global à notre application grâce à Redux ! Quelques manipulations sont nécessaires et nous allons pouvoir avoir des variables globales utilisables dans toute l'application sans pour autant les passer en paramètre.

Avec Redux, un concept est à comprendre, la vue appelle une action qui va être dispatchée au Store Cette action est directement appelée dans le code lorsqu'un utilisateur clic sur un bouton par exemple. Son but va être développé dans un Reducer, et c'est lui qui va traiter l'information et la mettre dans notre state global, le Store.

Ce que fait React Native derrière tout ça, c'est récupérer un state et le mettre dans une props.
À partir de là, nous allons pouvoir récupérer notre variable stockée dans le store à l'aide de cette ligne de code :

```
this.props.[variable]
```

Vous devez connecter votre composant au state global (mapStateToProps) et le tour est joué, vous avez accès à votre variable partout dans votre code.

Conclusion

L'écosystème des technologies autour du mobile ne cesse de grandir et évoluer. Les applications mobiles doivent intégrer toujours plus de fonctionnalités (Touch ID, Face ID, etc.) sur des supports eux aussi évoluant : écrans à encoche (iPhone X, Honor 10, etc.), écrans pliables, etc.

Il n'existe pas qu'une seule et même techno qui répond à l'ensemble des besoins.

Un certain nombre de facteurs vont entrer en ligne de compte dans votre choix, notamment :

- L'importance de la performance et du rendu ;
- L'enveloppe que vous avez ;
- Les compétences de vos développeurs en interne ;
- Devez-vous porter l'application sur le web ;
- Etc.

Comme vous avez pu le voir nous nous sommes attardés sur le Framework React Native au détriment des autres. Pourquoi ?

Les besoins couverts par React Native correspondent à 80% des projets de nos clients. Avoir une application la moins chère possible, de bonne facture et dans des délais optimaux.

React Native est un excellent compromis pour déployer une application de qualité tout en réduisant les coûts. Cependant, il existe d'autres technos similaires, notamment Flutter ou bien NativeScript, elles n'ont pas autant d'ancienneté et ont une communauté plus réduite, mais sont en pleine croissance, des technos à suivre.

À PROPOS

TheCodingMachine
Spécialisée depuis 2005 dans le développement Open Source, nous assurons l'ensemble des projets qui sont au cœur de votre stratégie digitale.

Nous intervenons depuis la mise en place jusqu'à la livraison (et même au-delà) en nous adaptant à vos besoins que ce soit en mode Agile ou au Forfait.

Hugo Averty est le directeur de l'agence lyonnaise.

Diplômé d'un Master en gestion de projet mobilité et sécurité du numérique. Intrapreneur passionné de nouvelles technos, d'UX avec une vision marketing business, il accompagne les entreprises dans leurs projets web et mobiles.

Aurélien Mutin est responsable technique de TheCodingMachine Lyon.

Titulaire d'un DUT Informatique et d'un titre d'Expert en Informatique et Systèmes d'Information SUPINFO, il débute sa carrière en tant que chef de produit pour une agence lyonnaise spécialisée dans le développement de plateforme destinées aux professionnels du tourisme.

Fort de 7 années d'expérience dans le développement web, Aurélien s'est spécialisé depuis 2017 dans le développement d'applications mobiles en React Native ainsi que dans la mise en place d'API complexes.

+33 (0)1 85 08 34 98
communication@thecodingmachine.com

56 rue de Londres
75008 Paris

TheCodingMachine
TCM://