Développement d’une application mobile d’échange de parking, quel Framework choisir ?

Travail de Bachelor réalisé en vue de l’obtention du Bachelor HES

par :

Loïc Schupbach

Conseiller au travail de Bachelor :

Rolf Hauri, professeur HES

Haute École de Gestion de Genève, 1 septembre 2018

Haute École de Gestion de Genève (HEG-GE)

Filière IG

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l’examen final de la Haute école de gestion de Genève, en vue de l’obtention du titre « Bachelor of Science HES-SO en Informatique de gestion ».

L’étudiant atteste que son travail a été vérifié par un logiciel de détection de plagiat.

L’étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J’atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Thônex, le 01.09.2018

Loïc Schupbach

[SIGNATURE]

Remerciements

Si vous avez des remerciements à formuler, à l’entreprise ou à toute autre personne qui a pu vous aider dans la réalisation du travail.

Les remerciements sont rédigés dans le style « **Corps de texte** ».

Résumé

Il est toujours difficile, lorsque l’on a une idée d’application, de savoir quels outils utiliser pour concrétiser cette idée. De nos jours, il est devenu essentiel de fournir notre application sur les plus grandes plateformes existantes (c-.à-d. IOS et Android) mais il n’est pas concevable d’adapter plusieurs fois l’application, une pour chaque plateforme.

C’est là qu’apparait les Frameworks de développement mobile. Ils permettent de créer une seule application dans un langage commun unique (souvent Javascript / HTML) puis de transpiler ces applications dans un langage qui permet de la lancer sur chacun des système mobiles précédemment cités.

Ces Frameworks sont très pratiques mais nécessitent une certaine connaissance de leurs fonctionnement pour pouvoir faire ce que nous voulons. Malheureusement il est impossible de tous les connaitre par cœur. Il faut donc faire un choix et faire tout le développement de notre application sur le Framework sélectionné.

En expliquant les Frameworks les plus connus / utilisés, j’espère que vous pourrez plus facilement comprendre leurs fonctionnement et choisir celui qui correspond le plus à vos attentes.

Table des matières

[Déclaration i](#_Toc522045551)

[Remerciements ii](#_Toc522045552)

[Résumé iii](#_Toc522045553)

[Liste des tableaux vi](#_Toc522045554)

[Liste des figures vi](#_Toc522045555)

[1. Introduction 1](#_Toc522045556)

[2. Etude des Frameworks existants 2](#_Toc522045557)

[2.1 Ionic 4](#_Toc522045558)

[2.1.1 Architecture 5](#_Toc522045559)

[2.1.1.1 Apache Cordova 5](#_Toc522045560)

[2.1.1.2 AngularJs 6](#_Toc522045561)

[2.1.1.3 Ionic 7](#_Toc522045562)

[2.1.2 Avantages 9](#_Toc522045563)

[2.1.3 Faiblesses 10](#_Toc522045564)

[2.2 Xamarin 10](#_Toc522045565)

[2.2.1 Architecture 10](#_Toc522045566)

[2.2.2 Avantages 11](#_Toc522045567)

[2.2.3 Faiblesses 11](#_Toc522045568)

[2.3 React Native 12](#_Toc522045569)

[2.3.1 Architecture 12](#_Toc522045570)

[2.3.2 Avantages 14](#_Toc522045571)

[2.3.3 Faiblesses 14](#_Toc522045572)

[2.4 Comparaison des Frameworks 15](#_Toc522045573)

[2.4.1 Comparaison 15](#_Toc522045574)

[2.4.2 Résumé 18](#_Toc522045575)

[3. Etude de l’application 19](#_Toc522045576)

[3.1 Fonctionnalités de l’application 19](#_Toc522045577)

[3.2 Besoins techniques de l’application 19](#_Toc522045578)

[4. Choix du Framework 20](#_Toc522045579)

[4.1 Analyse détaillé du choix 20](#_Toc522045580)

[4.2 Concordance avec l’application 22](#_Toc522045581)

[5. Implémentation de l’application 23](#_Toc522045582)

[5.1 Use-case 23](#_Toc522045583)

[5.2 Modèle de données 23](#_Toc522045584)

[5.3 Choix du système back-end 23](#_Toc522045585)

[5.4 Prototypage 23](#_Toc522045586)

[6. Développement de l’application 23](#_Toc522045587)

[6.1 Apprentissage du Framework 23](#_Toc522045588)

[6.2 Environnement de développement 23](#_Toc522045589)

[7. Rapport de test 23](#_Toc522045590)

[8. Conclusion 23](#_Toc522045591)

[Bibliographie 24](#_Toc522045592)

Liste des tableaux

[Tableau 1 – Résumé simple des Frameworks 15](#_Toc522045535)

[Tableau 2 - Résumé des performances des Frameworks 15](#_Toc522045536)

[Tableau 3 - Résumé des aspects de développement des Frameworks 16](#_Toc522045537)

[Tableau 4 - Résumé des aspects techniques des Frameworks 16](#_Toc522045538)

[Tableau 5 - Résumé des Frameworks 18](#_Toc522045539)

[Tableau 6 - Matrice de préférence 20](#_Toc522045540)

[Tableau 7 - Pondération des critères 21](#_Toc522045541)

[Tableau 8 - Analyse multicritère 22](#_Toc522045542)

Liste des figures

[Figure 1 - Frameworks, Librairies et Outils les plus populaires en 2018 3](#_Toc522045543)

[Figure 2 - Frameworks, Librairies et Outils les plus aimés en 2018 3](#_Toc522045544)

[Figure 3 - Architecture d'Apache Cordova 5](#_Toc522045545)

[Figure 4 - Architecture d'AngularJs 6](#_Toc522045546)

[Figure 5 - Structure de fichiers d’un projet Ionic 7](#_Toc522045547)

[Figure 6 - Structure de fichiers d'un projet AngularJS 8](#_Toc522045548)

[Figure 7 - Architecture de Xamarin 11](#_Toc522045549)

[Figure 8 - Architecture de React Native 13](#_Toc522045550)

# Introduction

# Etude des Frameworks existants

Le marché actuel propose pleins de Frameworks différents permettant de créer des applications mobiles. La plupart sont « cross-platform », c’est-à-dire qu’ils offrent la possibilité de créer un seul projet qui peut ensuite être utilisé sur plusieurs systèmes d’exploitation mobiles (IOS et Android, le plus souvent).

Pour permettre ce système « cross-platform », ces Frameworks n’utilisent pas le langage natif du téléphone (Java pour Android et Objective-C pour IOS) car il serait alors nécessaire de créer un projet pour chaque langage de programmation et donc pour chaque système d’exploitation mobile. Les Frameworks utilisent un langage différent des langages natifs (souvent Javascript pour la partie logique et HTML pour la partie visuel). Évidemment, ces langages ne sont pas compris par les téléphone mobiles et doivent être exécuté dans un environnement spécifique : le « Runtime Environnement ». Ces environnement encapsulent les projets « cross-platform » dans une « boite » qui comprend le langage du Framework. Cette boite offrent des entrées et des sorties permettant ensuite au Framework de communiquer directement avec les composants du téléphone (RAM, CPU, GPU, etc.). L’environnement de runtime peut différer selon les Frameworks. La partie visuel, quand à elle est le plus souvent affichée dans une WebView. Une WebView est un fenêtre prenant toute la place de l’écran et qui comprend les langages Web (HTML / CSS). Cette dernière est donc affichée à l’écran et la partie visuel de notre application est par la suite « collée » sur cette WebView.

Chaque langage possède son propre environnement de runtime. La WebView quant à elle est propre à chaque système d’exploitation mobile. Chaque plateforme propose nativement une WebView différente plus ou moins performante.

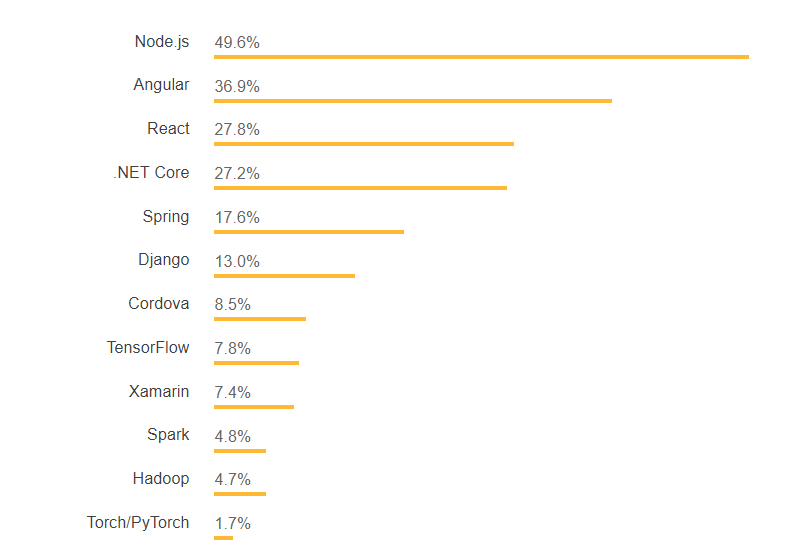
Il faut faire attention de bien vérifier si les fonctionnalités visuels que nous mettons en place sont compatibles avec la WebView native du système d’exploitation cible.

Comme expliqué précèdemment, il existe un grand nombre de Frameworks différents offrant la possibilité de créer une application « cross.platform ». Pour définir lequel est le plus adapté à la création d’une application d’échange de place de parking, je vais expliquer les avantages et les désavantages des 3 Frameworks les plus connus et utilisés.

Pour faire ce choix, je me suis basé sur un sondage réalisée par le site internet « Stack Overflow » (Stack Overflow, 2018).

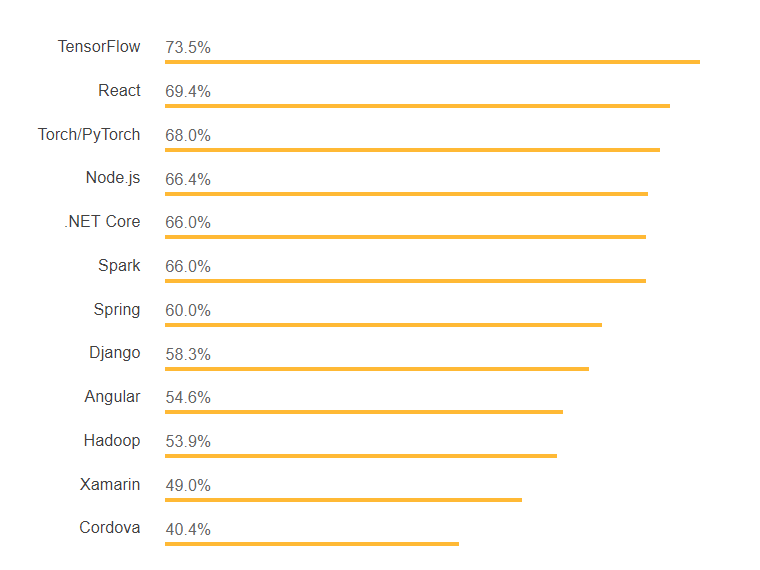
Ce sondage, réalisé pendant le mois de janvier 2018 a été répondu par plus de 67 000 développeurs jugés « qualifiés ». Les graphiques suivants représentent l’utilisation des Frameworks :

Figure 1 - Frameworks, Librairies et Outils les plus populaires en 2018



(Stack Overflow, 2018)

Figure 2 - Frameworks, Librairies et Outils les plus aimés en 2018



(Stack Overflow, 2018)

Ces deux graphiques représentent très bien la situation actuelle. Sur tous les Frameworks présents, 4 sont des Frameworks permettant le développement mobiles cross-platform triés par ordre de popularité :

* React (et sa version de développement mobile React Native)
* Angular (et sa version de développement mobile Ionic)
* Xamarin
* Cordova

Ces quatre Frameworks sont les plus populaire auprès de la communauté des développeurs en 2018. Je vais donc expliquer le fonctionnement de « React Native », « Ionic » et « Xamarin »

## Ionic

Ionic est un Framework utilisant les technologies WEB (Javascript et HTML) permettant la création de site Web et d’application mobile cross-platform. Ce Framework est une surcouche du Framework AngularJs permettant le binding de donnée (explication détaillée à la [section AngularJs](#_AngularJs)).

Il offre des composants graphiques déjà créés qu’AngularJS ne propose pas. De plus, l’architecture des dossiers et des fichiers est beaucoup plus structurée. Un système de thème a aussi été ajouté, permettant de créer des règles de styles générales à tout le projet.

L’objectif de ce Framework est d’offrir un développement court, efficace et ne nécessitant pas de grande connaissances dans le domaine. Le développement WEB étant la première chose que tout informaticien apprend, il est très facile de se lancer dans la création d’une application utilisant le Framework Ionic.

Ionic ne peut donc pas être utilisé seul. Il a besoin de fonctionner en corrélation avec AngularJs (pour le binding de données) ainsi qu’Apache Cordova (pour transformer notre projet en un package que le téléphone mobile peut comprendre).

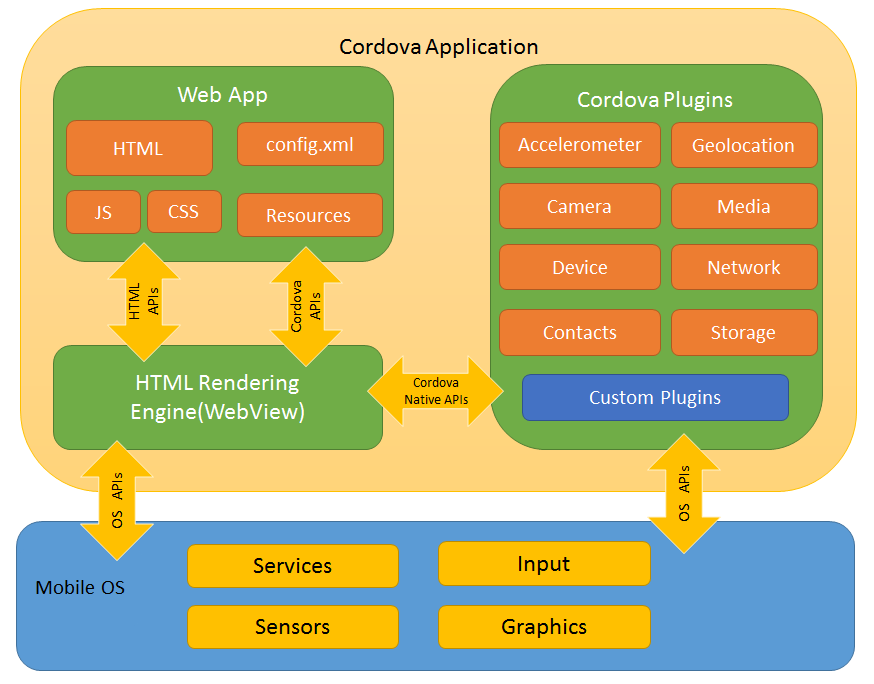
### Architecture

Comme décrit précédemment, Ionic est une surcouche du Framework AngularJs qui, lui-même, utilise le Framework Apache Cordova pour la création de l’APK. Je vais donc vous expliquer le fonctionnement de ces 3 Frameworks.

#### Apache Cordova

La création de l’APK et le test de notre application sur le téléphone mobile est une étape essentiel du développement. Il existe plusieurs façons de créer un APK. Apache Cordova propose d’encapsuler facilement notre application dans une package lisible par les systèmes d’exploitation mobiles et offre des accès aux composants natifs du téléphone (appareil photo, accéléromètre, etc.) via une collection de plugins (qui font office d’interface) intégrés à Apache Cordova.

Figure 3 - Architecture d'Apache Cordova



(Cordova, 2012)

Le schéma ci-dessus représente bien l’architecture de ce Framework.

Notre application, qui peut être créée avec n’importe quel Framework compatible, sera encapsulé dans la « boite » Web App. Cette dernière discute avec la WebView grâce à des systèmes d’Apis IN / OUT pour afficher le rendu visuel. De son côté, la WebView à un accès aux composants natifs du téléphone via les plugins d’Apache Cordova. Il est donc possible de récupérer des informations natives et de les traiter. Ces informations seront récupérées / calculées sur les composants natifs qui seront ensuite envoyées aux plugins d’Apache Cordova. Ceux-ci vont transpiler les données en quelque chose de lisible par notre application et les envoyer à la WebView qui, elle-même, transmettra les informations à la partie logique de notre application.

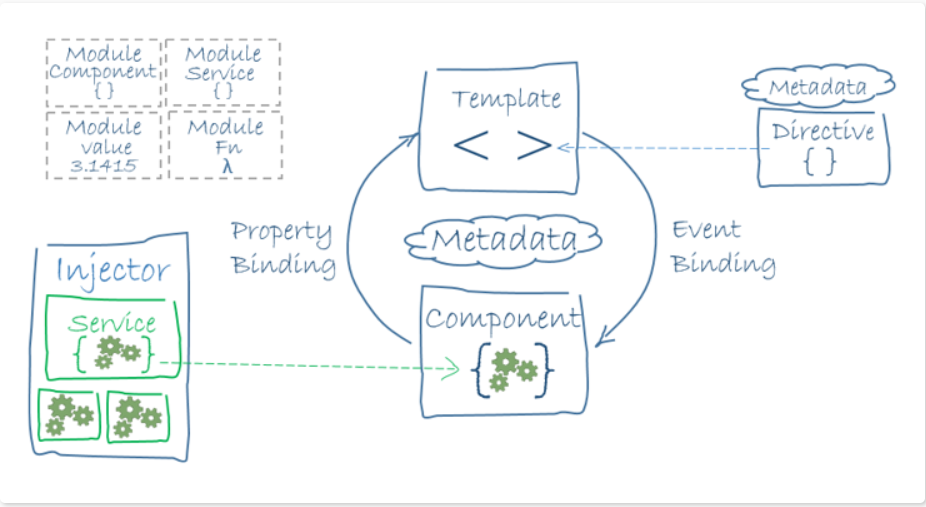
Apache Cordova utilise l’environnement de runtime natif JavaScript de chaque système d’exploitation pour fonctionner l’application et créer les apks.

#### AngularJs

AngularJs est un Framework offrant la possibilité de créer des pages WEB et des applications mobiles de manière simplifié grâce à des systèmes de déclaration de modèles, d’injection de dépendances et de binding de données.

AngularJs va relier de manière simplifié les données de la partie logique à l’affichage et inversement.

Figure 4 - Architecture d'AngularJs



(Angular, 2009a)

On peut voir sur ce schéma le fonctionnement d’AngularJs. Chaque objet logique de notre site Web / application (page, liste, bouton personnalisé, etc.) va être enregistrée en tant que « composant ». Un composant désigne la partie logique d’une page et est donc reliée à un modèle écrit en HTML / CSS. Cette liaison permet de faire du « Property Binding » et de « l’Event Binding ».

Le Property Binding permet de mettre à jour automatiquement les données de notre application. C’est-à-dire que quand une donnée est affichée sur le modèle et que la variable représentant cette donnée change dans le composant, l’affichage de cette donnée va automatiquement être mis-à-jour dans le modèle. Ce système fonctionne aussi en sens inverse. Lorsque l’utilisateur va écrire du texte dans un champ de texte, il est possible d’enregistrer le contenu de ce champ en temps réel dans une variable du composant.

L’Event Binding quand à lui permet de déclencher une certaine méthode de la partie logique quand l’utilisateur déclenche une certaine action de la partie visuel de notre application.

AngularJs propose aussi un système d’injection de dépendance. Cela permet d’ajouter dans un composant un lien sur un autre composant ou un autre service (Accéléromètre, Network, etc.) et de pouvoir accéder aux données et fonctions de ce dernier.

#### Ionic

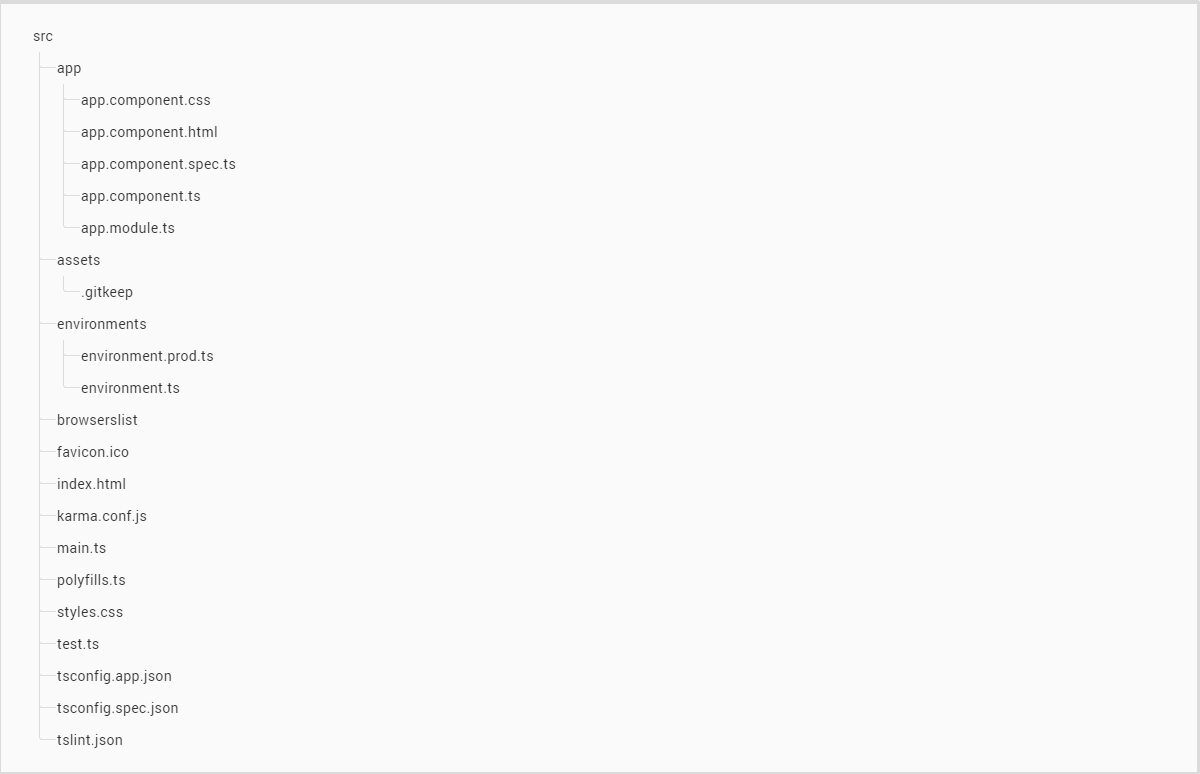
Ionic n’est qu’une surcouche à AngularJs. Il est là uniquement pour simplifier la structuration de notre projet et offrir une stylisation de l’application plus poussée qu’AngularJs.

Figure 5 - Structure de fichiers d’un projet Ionic



(Drifty, 2017)

Figure 6 - Structure de fichiers d'un projet AngularJS



(Angular, 2009b)

On peut constater que la structure de dossier d’une projet Ionic est clairement plus structurée que celle d’un projet AngularJS. Le dossier « src » ne contient que les éléments à modifier contrairement au dossier « src » d’AngularJs qui contient plusieurs fichiers de configuration. De plus, le dossier « src » d’Ionic contient des sous-dossier structurant le projet avec des pages, des assets et différents thèmes.

De plus, Ionic à inclue dans son Framework plusieurs composants graphiques responsive ainsi que cross-platform facilement utilisables.

Il est aussi possible d’utiliser le langage TypeScript à la place du langage JavaScript pour le développement avec le Framework Ionic. TypeScript est un langage basé sur le JavaScript (une sorte de surcouche) offrant le typage, les interfaces, les génériques, les namespaces ainsi que un contrôle des erreurs plus poussé.

### Avantages

Ionic est un Framework récent permettant le développement simple et efficace d’application cross-platform. C’est dans sa structuration et son fonctionnement simplifié que réside sa plus grande force.

Le développement cross-platform devenant la norme dans la création d’application, cela en fait un atout majeur pour la Framework Ionic. C’est tout à fait compréhensible quand on sait que le développement cross-platform permet :

* Une réduction des coûts de développement et de maintenance
* Des connaissances nécessaires réduites
* La création de code réutilisable
* Une migration et maintenabilité simplifié

Comme Ionic utilise le JavaScript ou TypeScript ainsi que l’HTML pour la partie visuel, cela permet à un grand nombre de développeurs d’utiliser le Framework. En effet, les langages WEB sont les premiers langages que tout développeur apprend car ils sont très simple de compréhension. Il est très facile de trouver de l’aide ou des tutoriels sur internet.

Un autre grand atout d’Ionic est sa portabilité. Il est effectivement possible de créer un site web responsive utilisant les composants donnés par Ionic et ensuite de transformer ce site web en une application en seulement quelques modifications.

Ionic offre en effet une portabilité tellement grande avec les composants qu’il offre qu’il est même possible de créer un seul code pour une application Android, une application IOS et un site web (pour autant qu’il n’y ait pas de PHP). Seul quelques pourcents du code doit être adapté aux plateformes pour permettre sont fonctionnement (cela concerne surtout les plugins qui ne fonctionnement pas de la même manière sur toutes les plateformes).

Un point important d’Ionic est son système de débugge. Comme l’application peut être lancée dans un navigateur, il est possible d’afficher des informations dans la console et de modifier l’interface utilisateur en temps réel.

### Faiblesses

Actuellement, l’expérience utilisateur est au centre de toutes les préoccupations dans l’informatique. Que ce soit pour des clients lourds, riches, léger, des applications de bureau ou des applications mobiles, il est important que l’utilisateur ait un ressenti positif de l’application.

Un point qu’Ionic ne parvient pas toujours à combler. En effet, comme Ionic est en fait composé de 3 couches ( Ionic 🡺Angular 🡺 Apache Cordova), les performances de l’application sont grandement diminuées. En cas d’utilisation gourmande de l’application, le nombre d’images par seconde va grandement chuter et l’utilisateur aura l’impression que l’application fonctionne au ralenti et qu’elle n’est pas fluide.

Un autre gros problème de ce Framework est la compatibilité entre les différents « sous-frameworks » qui le compose. Il est assez fréquent que des changements soient faits au niveau d’Apache Cordova (surtout au niveau des plugins) et que ces changements cassent le fonctionnement actuel de notre application. De plus, certains plugins ne sont pas fonctionnels et ne peuvent pas être utilisé lors du passage en production de l’application. (Vincent, 2016)

## Xamarin

Xamarin est un Framework dédié au développement d’application mobile « cross-platform » (Android, IOS et pour ceux qui veulent, Windows Phone). Contrairement à la plupart des autres Frameworks de développement cross-platform, Xamarin est basé sur le Framework .NET et utilise donc le langage C# et pas JavaScript.

Contrairement aux pensées, Xamarin n’est pas un Framework interprété mais compilé nativement, ce qui lui donne la possibilité de créer des applications très performantes avec un design proche d’un design natif. (Altexsoft, 2018a)

Ce Framework offre la possibilité de créer un seul code source logique codé en C# et transférable sur plusieurs OS cibles. En ce qui concerne la partie visuel de l’application, il existe deux options :

* Xamarin.Android et Xamarin.IOS (ou encore Xamarin.Windows)
* Utilisation de Xamarin.Forms permettant de créer un seul design « cross-platform » pour Android, IOS et Winows Phone.

La première option est bien évidemment la plus performante. En effet, chaque OS cible aura son propre design créer en langage natif et sera donc très performant. De plus, créer un design en langage natif nous permet de rester le plus fidèle possible aux normes des interfaces utilisateurs proposées par chaque plateforme. Le désavantage réside dans l’obligation d’écrire plusieurs fois le même code pour chaque OS ciblé.

La deuxième option, Xamarin.Forms, facilite grandement le développement et permet d’avoir un design unifié entre les différentes plateformes mais pose un problème de taille. Les performances seront grandement diminuées pour certaines opérations et la création de design compliqué sera impossible. (Altexsoft, 2018b)

### Architecture

Figure 7 - Structure simplifié d'une application Xamarin

Comme expliqué précédemment, Xamarin offre la possibilité de créer un seul code logique et une ou plusieurs interfaces utilisateurs.

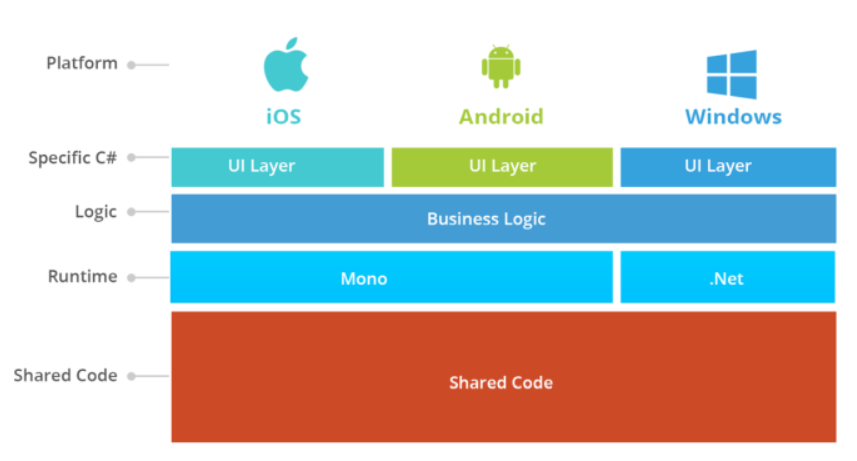
L’image ci-contre représente la structure simplifiée d’une application Xamarin. Toute la partie logique et accès aux composants natifs du téléphone se trouve dans un bloc de codé partagé et unique pour toutes les plateformes. Le seul code non partagé est le design de l’interface, pour autant qu’il ne soit pas créé avec Xamarin.Forms.

(Altexsoft, 2018a)

Tout comme pour Ionic, les systèmes d’exploitations mobiles ne comprennent pas le langage C# et doivent donc avoir un environnement de runtime spécifique. Étant donné que ces plateformes n’ont pas cette environnement de manière native, Xamarin a pris la décision de l’ajouter dans le code source de l’application ainsi. De cette manière, quel que soit le système d’exploitation mobile sur lequel est lancé l’application, elle pourra fonctionner.

Le schéma ci-dessous représente l’architecture complète d’une application Xamarin, incluant l’environnement de runtime, le code partagé et les interfaces utilisateurs.

Figure 8 - Architecture de Xamarin



(Nexgendesign, Non daté)

En créant des interfaces différenciées pour chaque plateforme cible, les composants de ces interfaces utilisateurs vont être des composants natifs. Le seul code non natif sera alors le code partagé en C# qui tourne dans sont propre environnement de runtime créer pour avoir des performances maximales. Si l’on décide d’utiliser Xamarin.Forms, l’interface utilisateur ne sera alors plus composée de composants natifs et perdra alors grandement en performance. Il sera néanmoins possible de partagé de 90% à 100% de code en utilisant cette option.

### Avantages

Xamarin est un Framework proposant une approche totalement différentes des autres Frameworks de développement mobiles. Le fait qu’il utilise le langage C#, un langage fortement typé, y ajoute une couche de protection contre les éventuels bugs générés par un code approximatif.

Comme expliqué précédemment, les performances d’une application créée grâce à Xamarin est très proche des performances d’une application native. Cela en fait un atout majeur pour ce Framework. En effet, il est très important d’avoir une application rapide et fluide car les utilisateurs aiment de moins en moins attendre qu’une application charge.

La possibilité qu’offre Xamarin de créer l’interface utilisateur de 2 manières différentes est aussi un atout pour le Framework. Si l’application développé n’utilise pas beaucoup de ressources, il est possible de créer une seule interface utilisateur partagée sur chaque plateforme. Mais si l’application est une application lourde, Xamarin donne aussi la possibilité de créer des interfaces utilisateurs natives pour garder des performances proches du natif. Il est aussi possible d’utiliser Xamarin.Forms pour créer des prototypes de notre application est de les montrer aux clients. Le client donne sont avis et lorsque l’interface sera définitive, elle peut être réécrite pour chaque OS cible et donc augmenter les performances de l’application.

### Faiblesses

L’utilisation de Xamarin oblige le développeur a se former à deux nouveaux langages : le C# et le langage qu’utilise Xamarin pour créer ces interfaces Xamarin.Forms, le XAML.

La communauté utilisant le Framework Xamarin est aussi bien plus faible que celle utilisant les Frameworks utilisant les technologies WEB. Il sera donc plus difficile de trouver des réponses à nos questions lorsque nous aurons un problème.

Comme expliqué précédemment, l’utilisation de Xamarin.Forms réduit grandement les performances d’une application créée avec Xamarin. Il n’est donc pas possible de créer des interfaces utilisateurs compliquées ou contenant des graphismes lourds.

La taille des applications est aussi une inconvénient de Xamarin par rapport à ces concurrents. Comme une application Xamarin n’est pas comprise par le système d’opération du téléphone mobile, l’application emporte avec elle d’autres outils pour la faire fonctionner (runtime, assemblage de libraires de classes de base, etc.)

Figure - Taille d'une application "Hello World" avec Xamarin



(Altexsoft, 2018a)

Xamarin ayant été acheté par Microsoft, ce dernier a rendu l’utilisation de cet outil gratuit mais l’a aussi intégré à son IDE : Visual Studio.

Il est donc nécessaire d’obtenir Visual Studio pour créer une application Xamarin. Heureusement pour nous, une version gratuite de Visual Studio existe mais elle n’offre pas toutes les fonctionnalités qui nous permettent de travailler correctement sur notre projet.

Pour profiter pleinement de Visual Studio et de Xamarin, il est essentiel d’acquérir une licence Visual Studio qui coute extrêmement cher (plus de 1000 dollars)

## React Native

React Native est un Framework de développement mobile basé sur le Framework ReactJs pour créer et composer des interfaces riches à partir de composants déclaratifs codés en JavaScript.

React Native se distingue des autres Frameworks de développement mobile utilisant les technologies WEB car il ne crée par une interface tournant dans une WebView (comme Ionic) mais en proposant une système semblable à Xamarin qui permet la création d’interfaces utilisateurs natives. React Native va, en quelque sorte, encapsuler les composants natifs dans des composants compréhensibles pour le Framework. (React Native, 2018)

Ce système impose à React Native de créer les interfaces dans un langage différent du HTML. C’est pour cela que ce Framework utilise un langage différent du HTML pour la création de ces interfaces utilisateurs. Ce langage ressemble fortement à l’HTML mais ne représente pas les mêmes objets. Chaque balise est une encapsulation d’une élément natif du téléphone. Ce système a été choisi car React par du principe que la logique du rendu est fortement couplée à la logique d’affichage des interfaces utilisateurs : comment les évènements sont traités, comment l’état de l’application change au fil du temps et comment les données sont préparées pour l’affichage. (React, 2018)

### Architecture

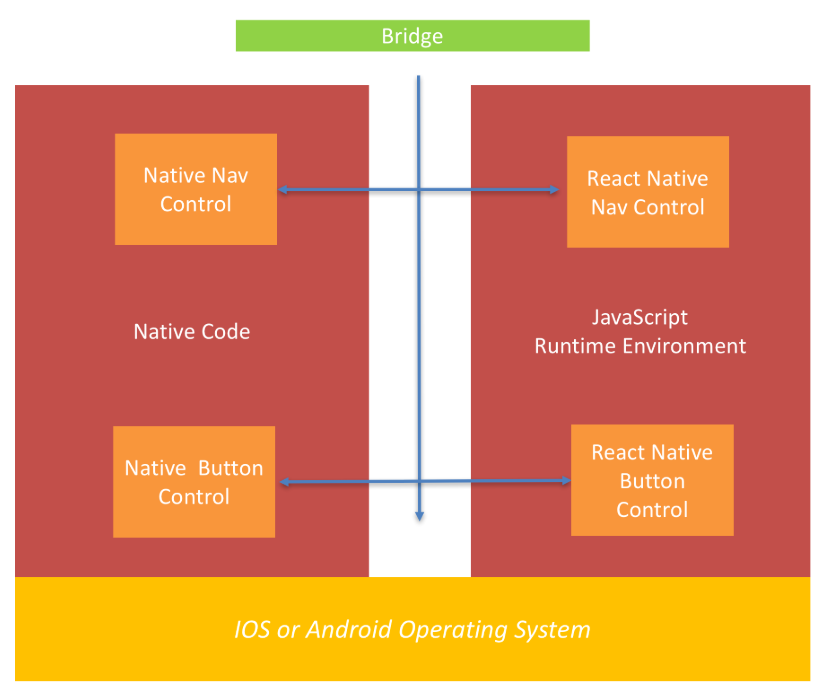
React Native à un fonctionnement très spécifique à lui-même. En effet, un Framework basique tel que Ionic va avoir ses propres composants qui seront connectés via des APIs aux composants natifs du téléphone. Une multitude de couches se superposent alors pour offrir l’accès aux fonctionnalités du téléphone.

React Native a été construit d’une toute autre manière. Chaque composant est directement branché au composant natif dont il fait la référence. Aucune couche supplémentaire n’est nécessaire pour communiquer entre les composants natifs et ceux offerts par React Native.

Il est néanmoins important de comprendre que l’on n’interagit pas immédiatement avec les composants natifs du téléphone. React Native reste un Framework utilisant le langage JavaScript et ne peut donc pas directement être lancé sur un téléphone Android ou IOS.

C’est pour cela que React Native utilise un environnement de runtime spéciale à Javascript.

Figure 10 - Architecture de React Native

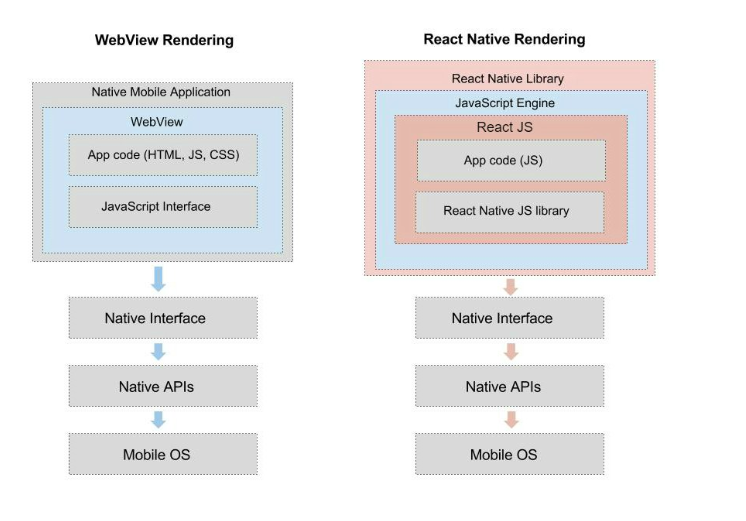


(Heard, 2017)

L’application est alors lancée dans cet environnement, ce qui permet d’accéder au langage JavaScript et de le faire tourner. Des ponts sont ensuite créés entre les composants offerts par React Native et les composants natifs du téléphone. Seuls les composants sont donc connectés au téléphone. Le reste de l’application communique ensuite uniquement via les composants React Native qui sont eux même connectés aux composants natifs du téléphone.

L’interface créée via le Framework React Native n’est donc pas une interface « HTML/CSS ». L’interface est semi-native et est lancée directement sur la couche native du téléphone. React Native n’a pas besoin d’une WebView pour afficher l’application. La disparition de la couche WebView ainsi que de la couche d’accès aux composants augmente grandement les performances d’une application React Native.

Figure - Différence WebView et React Native



(Altexsoft, 2018)

On peut voir sur cette image que React Native ne dispose pas de WebView. L’interface utilisateur, qui est dans la WebView pour une application JavaScript / HTML, est ici directement dans la boite « Native interface ».

### Avantages

Le point important à retenir du Framework React Native est la manière de lier le Framework aux composants Natifs du téléphone. Cela enduit des performances accrues, très proches d’une application native tout en gardant un développement cross-plateforme (un seul langage pour chaque plateforme). Evidemment, chaque système d’exploitation a ces spécificités et il sera nécessaire d’adapter légèrement son code pour chaque plateforme ciblée. De 90% à 100% du code créer pourra être partagé entre toutes les plateformes.

Il est possible également de créer ses propres composants natifs (en Objective-C ou Java) et de les implémenter directement dans le Framework React Native. Comme le Framework utilise des ponts pour communiquer, il est très simple de rajouter ces propres composants.

Évidemment, toute la partie logique de l’application étant codée en JavaScript, aucun apprentissage devra être fait pour apprendre ce langage qui est très connu.

Un point important de React Native est son système de débugg. Il est possible de lancer son application directement sur son téléphone est de pouvoir débugger en tant réel les problèmes depuis son ordinateur. Ce système est très semblable à celui d’Ionic.

### Faiblesses

Comme React Native est basé sur le Framework ReactJS, une connaissance de ce dernier est conseillée.

Comme React Native créer une encapsulation des composants natifs du téléphone, il ne propose qu’une petite collection de composants. Pour créer des éléments compliqués, il faudra faire un empilement de ces composants qui peut vite se révéler compliqué.

Le plus gros désavantage réside dans l’ajout de composants externes / tierces à notre application. La communauté à créer pleins de composants et modules augmentant les possibilités de création de React Native. Malheureusement, ces composants ne sont pas toujours mis à jour et souvent, certains composants ne vont pas pouvoir être implémenté à notre projet.

## Comparaison des Frameworks

Chaque Framework comporte des avantages et des désavantages. Une application créée via le Framework Ionic pourra être très personnalisée mais sera vite limitée au niveau des performances. Xamarin est un juste milieu avec des bonnes performances, un accès aux composants natifs simplifiés mais une création d’interface limitée et l’obligation de connaitre le langage C#. React Native propose un mélange des 2 autres Frameworks avec des performances proches des performances natives, un développement simplifié offrant le cross-plateforme mais une personnalisation moyenne.

### Comparaison

Les quelques graphiques et tableaux ci-dessous résument les points importants :

Tableau 1 – Résumé simple des Frameworks

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ionic | Xamarin | React Native |
| Langages | JavaScript / TypeScript et HTML | C# et XAML | JavaScript et composants natifs |
| Accès aux composants natifs direct | Non | Oui | Oui |
| Idées générales | Utilise les technologies Web pour une portabilité maximale | Reste le plus proche possible du natif | Approche fonctionnelle : l’interface est une fonction de l’état |

(Cruxlab, 2017)

Tableau 2 - Résumé des performances des Frameworks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Ionic** | **Xamarin** | **React Native** |
| Compilation du code | IOS | Interpréteur JIT avec plugins | AOT | Interpréteur AOT |
| Android | JIT | JIT | JIT |
| Support du 64-bit | IOS | Oui | Oui | ? |
| Android | ? | Oui | Non |
| UI | IOS et Android | HTML | Composants natifs | Composants natifs |

(Cruxlab, 2017)

Tableau 3 - Résumé des aspects de développement des Frameworks

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **Outils officiels** | **Ionic** | **Xamarin** | **React Native** |
| Lancement manuel |  | Oui | Oui | Oui | Oui |
| Lancement automatique |  | Non | Oui | Non | Oui |
| Permutation du code à chaud | IOS | Non | Non | Non | Oui |
| Android | Oui | Non | Non | Oui |
| Permutation du code à froid | IOS | Non | Oui | Non | Oui |
| Android | Oui | Oui | Oui | Oui |
| Développement dans le navigateur |  | Non | Oui (aspect différent !) | Non | Non |
| Mise-à-jour instantanées |  | Non | Oui | Non | Oui |

(Cruxlab, 2017)

Tableau 4 - Résumé des aspects techniques des Frameworks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Ionic** | **Xamarin** | **React Native** |
| Capacités de liaison de code | Utiliser les classes Java, Objective-C et @objc Swift à partir d'un Framework multi-plateforme | Nécessité d’avoir un adaptateur | Toutes les libraires binaires | Nécessité d’avoir un adaptateur |
| Utilisation de classes multi-plateformes à partir d'un code traditionnel | Non | Compliqué mais faisable | Oui |
| Liaison a des bibliothèques standards | Set de fonctionnalités supportés | Partiel | Complet | Partiel |
| Type de liaison | Couche d’abstraction | One-to-one | Couche d’abstraction |
| UI | Set complet de composants disponible | Non | Oui | Non |
| Design ressemblant au design natif | Oui | Partiel | Partiel |
| Exécution du code en tâche de fond | IOS | Uniquement prévention de sommeil du téléphone | Semblable à du code natif | Non |
| Android | Uniquement prévention de sommeil du téléphone | Semblable à du code natif | Headless JS |

(Cruxlab, 2017)

### Résumé

Tableau 5 - Résumé des Frameworks

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Ionic** | **Xamarin** | **React Native** |
| Performances | Mauvaises / Moyennes | Bonnes | Bonnes |
| Look natif | Oui | Partiel | Partiel |
| Outils de développement modernes | Partiel | Non | Oui |
| Fonctionnalités des OS | Partiel | Oui | Partiel |
| Connaissances nécessaires | Faibles | Hautes | Moyennes |

Inspiré de (Cruxlab, 2017)

# Etude de l’application

L’application vise à régler le problème du manque de place de parking dans la ville de Genève. Elle permettra à toute personne disposant d’une place de parking privée de la mettre en location et toute personne possédant l’application pourra alors louer cette place pour un certain temps.

## Fonctionnalités de l’application

L’application offrira les fonctionnalités suivantes :

* Enregistrement au sein de l’application
* Suppression de son compte
* Connexion à son compte
* Enregistrer une place de parking
* Visualiser toutes les places de parking actuellement disponibles dans un cercle de recherche sur une carte
* Louer une place de parking immédiatement
* Réserver une place pour une date définie
* Modifier ces informations de connexion et de profil
* Voir toutes ces places et réservations

## Besoins techniques de l’application

Cette application vise a être utilisée à tout temps. Il est donc essentiel qu’elle soit rapide, fluide et simple d’utilisation.

Une carte doit s’afficher pour voir les places disponibles (sous forme de markers). Cette dernière doit aussi offrir la possibilité d’afficher un cercle de rechercher pour voir toutes les places disponibles dans un certain périmètre.

Il est évidemment logique que l’application puisse être disponible sur Android (puis sur IOS pour la suite).

Il faut que l’application se lance rapidement et ait des performances proches d’une application native.

# Choix du Framework

Les trois Frameworks que je vous ait présenté sont très intéressants. Chacun a ses avantages et désavantages.

Après avoir analysé les fonctionnalités et besoins de l’application, j’ai estimé que React Native était le Framework le plus adapté. Il offre des performances accrues ainsi que des accès aux fonctionnalités natives complètes. De plus, beaucoup de libraires peuvent y être ajoutés pour offrir une expérience utilisateurs améliorée.

## Analyse détaillé du choix

Pour analyser dans les détails le choix du Framework, j’ai utilisé une analyse multicritère basée sur certains critères définis grâce à une matrice de préférence.

Voici les différents critères sélectionnés :

1. Connaissance du langage associé
2. Accès aux fonctionnalités natives du téléphone
3. Framework bien documenté
4. Performances
5. Maintenabilité

Tableau 6 - Matrice de préférence

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

4

5

4

1

4

1

2

4

2

2

Tableau 7 - Pondération des critères

|  |  |  |
| --- | --- | --- |
| Numéro | Critère | Pondération |
| 1 | Connaissance du langage associé | 2 |
| 2 | Accès aux fonctionnalités natives du téléphone | 3 |
| 3 | Framework bien documenté | 0 |
| 4 | Performances | 4 |
| 5 | Maintenabilité | 1 |

Évidemment, comme mon travail de Bachelor est un mix entre une explication des Frameworks et la création d’une application, il est essentiel de choisir un Framework proposant des performances importantes ainsi qu’un accès aux fonctionnalités natives facilité. Néanmoins, ne disposant pas d’un temps de travail illimité, il est tout aussi important que le langage associé au Framework soit un langage que je connaisse. Je ne dispose pas du temps pour apprendre les spécificité du Framework que je choisir et aussi un nouveau langage.

J’ai, pour un premier temps, décidé de créer mon application sur la plateforme Android pour une question de simplicité et de couts. Mais il est important de savoir que j’envisage, à terme, de fournir mon application sur Android et IOS. C’est pour cela que tous les Frameworks proposés sans cross-platform et donc que ce critère ne rentrait pas en compte dans le choix du Framework

Tableau 8 - Analyse multicritère

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Critères |  | Ionic | | Xamarin | | React Native | |
|  | **Pondération** | **Points** | **Val. Pond** | **Points** | **Val. Pond.** | **Points** | **Val. Pond.** |
| Connaissance du langage associé | 2 | 10 | 20 | 5 | 10 | 8 | 16 |
| Accès aux fonctionnalités natives du téléphone | 3 | 8 | 24 | 10 | 30 | 8 | 24 |
| Framework bien documenté | 0 | Critère supprimé | | | | | |
| Performances | 4 | 5 | 20 | 9 | 36 | 9 | 36 |
| Maintenabilité | 1 | 8 | 8 | 5 | 5 | 8 | 8 |
| Total | | **72** | | **81** | | **84** | |

## Concordance avec l’application

React Native est le Framework répondant le mieux aux fonctionnalités et besoins de mon application. L’analyse multicritère le prouve. Il est précédé par Xamarin qui est seulement à 3 points de React Native. En effet, ce Framework propose de bonnes performances ainsi qu’un accès aux fonctionnalités natives complet (qui sont les critères les plus importants) mais son langage ne concorde pas avec mes connaissances. Il perd donc beaucoup de points à ce niveau-là. De plus, la maintenabilité d’une application créée avec Xamarin n’est pas du tout évidente. En effet, on ne peut pas mettre à jour notre application tant que le Framework Xamarin n’a pas été mis à jour. Il embarque son propre environnement de runtime qui doit s’adapter à chaque mise-à-jour de la plateforme ciblée. Un point négatif pour Xamarin mais positif pour React Native et Ionic qui utilise un environnement de runtime JavaScript intégré à l’appareil et donc mit à jour en même temps que l’OS.

# Implémentation de l’application

## Use-case

## Modèle de données

## Choix du système back-end

## Prototypage

# Développement de l’application

## Apprentissage du Framework

## Environnement de développement

# Rapport de test

# Conclusion

Bibliographie

ALTEXSOFT, 2018a. Pros and Cons of Xamarin vs Native Mobile Development. In : [en ligne]. 8 mai 2018. [Consulté le 17 août 2018]. Disponible à l’adresse : https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/.

ALTEXSOFT, 2018b. Xamarin vs React Native vs Ionic: Cross-platform Mobile Frameworks Comparison. In : [en ligne]. 19 février 2018. [Consulté le 13 juin 2018]. Disponible à l’adresse : https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-cross-platform-mobile-frameworks-comparison/.

ANGULAR, 2009a. Angular - Architecture overview. In : [en ligne]. 2009. [Consulté le 18 juin 2018]. Disponible à l’adresse : https://angular.io/guide/architecture.

ANGULAR, 2009b. Angular. In : [en ligne]. 2009. [Consulté le 18 juin 2018]. Disponible à l’adresse : https://angular.io/.

REACT, 2018. Introducing JSX – React. In : [en ligne]. 2018. [Consulté le 19 août 2018]. Disponible à l’adresse : https://reactjs.org/docs/introducing-jsx.html.

ALTEXSOFT, 2018. Pros and Cons of ReactJS and React Native. In : [en ligne]. 10 mai 2018. [Consulté le 17 août 2018]. Disponible à l’adresse : https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/.

CORDOVA, Apache, 2012. Architectural overview of Cordova platform - Apache Cordova. In : [en ligne]. 2012. [Consulté le 15 juin 2018]. Disponible à l’adresse : https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

CRUXLAB, 2017. Xamarin vs Ionic vs React Native: differences under the hood. In : [en ligne]. 2017. [Consulté le 8 juin 2018]. Disponible à l’adresse : https://cruxlab.com/blog/reactnative-vs-xamarin/.

DRIFTY, 2017. Ionic Framework. In : *Ionic Framework* [en ligne]. 2017. [Consulté le 18 juin 2018]. Disponible à l’adresse : https://ionicframework.com/docs/cli/projects.html.

HEARD, Pete, 2017. React Native Architecture : Explained! In : *React Native Architecture : Explained!* [en ligne]. 13 juin 2017. [Consulté le 25 juin 2018]. Disponible à l’adresse : https://www.logicroom.co/react-native-architecture-explained/.

NEXGENDESIGN, Non daté. 7 Reasons Xamarin Can Be a Trouble | NexGenDesign. In : [en ligne]. Non daté. [Consulté le 19 juin 2018]. Disponible à l’adresse : http://www.nexgendesign.com/xamarin-troubles.

REACT NATIVE, 2018. React Native · A framework for building native apps using React. In : [en ligne]. 2018. [Consulté le 25 juin 2018]. Disponible à l’adresse : https://facebook.github.io/react-native/index.html.

STACK OVERFLOW, 2018. Stack Overflow Developer Survey 2018. In : *Stack Overflow* [en ligne]. 31 janvier 2018. [Consulté le 18 juin 2018]. Disponible à l’adresse : https://insights.stackoverflow.com/survey/2018/.

VINCENT, 2016. Cordova : Applications mobiles hybrides. In : *VinceOPS* [en ligne]. 25 février 2016. [Consulté le 18 juin 2018]. Disponible à l’adresse : https://vincent-g.fr/2016/02/25/ionic-apache-cordova-developpement-mobile-hybride/.