h e g

Haute école de gestion
Genève

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

Hes·so GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

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

Liste des tableaux

Liste des figures

1. Introduction

2. Etude des Frameworks existants

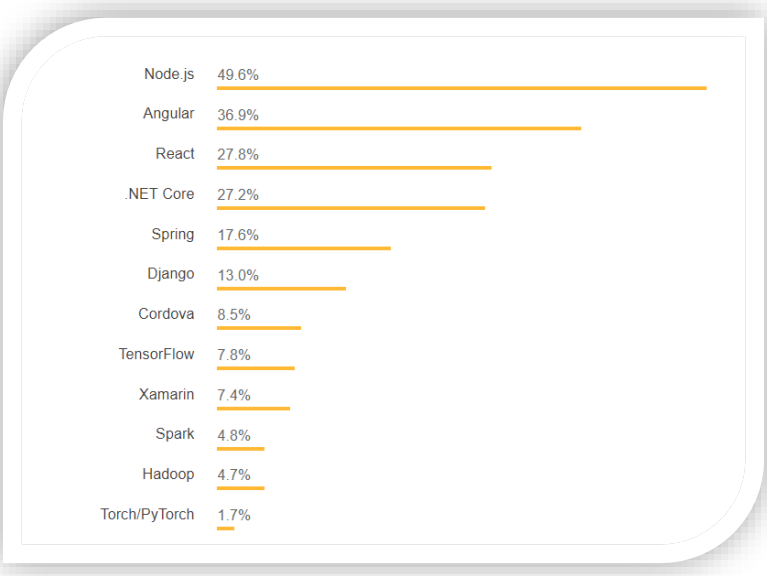
Plusieurs Frameworks existent actuellement sur le marché. La plupart utilisent le langage JavaScript (ou un langage dérivé de ce dernier) comme langage de programmation. Les ordres écrits dans ce langage ne sont pas directement compris par les éléments natifs du téléphone, qui ont leur propre langage (Objectif-c pour IOS et Java pour Android). Il faut alors une couche de liaison entre l'application et le téléphone pour permettre la bonne compréhension de l'application. C'est, dans la plupart des cas, cette couche de liaison qui diffère entre les différents Frameworks.

Vu le nombre de Frameworks existant, il n'est pas concevable d'expliquer le fonctionnement de chacun d'eux. J'ai donc choisi de développer plus en profondeur 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).

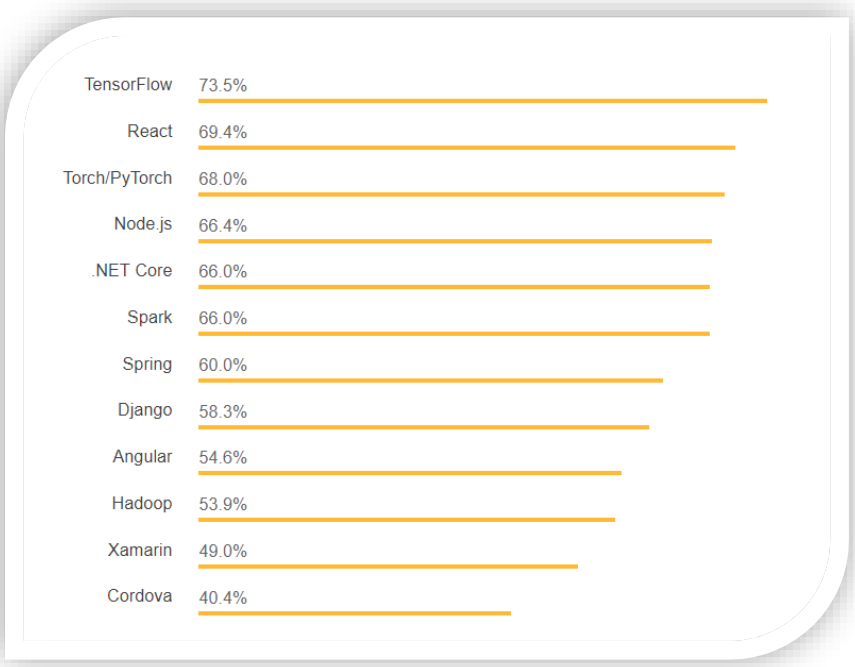
Ce sondage, réalisé pendant le mois de janvier 2018 a été répondu par plus de 67 000 développeurs jugés « qualifiés ». Les graphiques suivants représentent l'utilisation des Frameworks :

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)

Sur la base de ces 2 graphiques, il est très facile de remarquer les Framework les plus utilisés et appréciés. Sur les différents Frameworks cités dans les graphiques ci-dessus, seulement quelques un ont été créer dans le but de faire du développement mobile.

Voici la liste des Frameworks de développement mobile que l'on peut trouver dans les graphiques :

- React, dans sa forme mobile ➔ React Native
- Angular, dans sa forme mobile la plus utilisée ➔ Ionic
- Xamarin
- Apache Cordova

Les 4 Frameworks cités ci-dessus sont les 4 Frameworks les plus utilisés / appréciés par la communauté des développeurs en 2018. Il est néanmoins important de noter qu'Apache Cordova n'est plus utilisé, de nos jours, comme un Framework de développement mobile mais comme un Framework de transpilation pour lancer l'application sur le téléphone.

Je vais donc expliquer le fonctionnement de « React Native », « Ionic» et « Xamarin »

2.1 Ionic

Ionic est un Framework permettant le développement d'applications mobiles ou de site web en langage WEB (HTML/CSS). La partie logique d'un projet Ionic peut être gérée en JavaScript ou en TypeScript, qui est une surcouche au langage JavaScript, améliorant quelques aspect pratique au langage. L'objectif de ce Framework est d'offrir un développement plus court et plus efficace sur les différents OS mobiles existant en permettant de garder un design et des interactions aussi proches que sur une application native. (Kauderer, 2015). Ionic se base sur Angular pour permettre l'échange des données entre la vue et le code logique de l'application. Pour transpiler l'application en un package installable sur un téléphone mobile, Ionic utilise Apache Cordova. Ce dernier lui offre aussi la possibilité d'accéder aux fonctionnalités natives du téléphone, tel que l'appareil photo, le calendrier et bien d'autres encore.

Le but d'Ionic est d'offrir un accès simplifié aux fonctionnalités qu'offre Apache Cordova et de nous proposer des outils permettant la création d'une application (templates, objets déjà stylisés, etc)

2.1.1 Architecture

L'architecture basique d'une application Ionic est la suivante :

Figure 3 - Structure d'un projet Ionic

```
project/
├ ionic.config.json # Ionic project config file
├ package.json
├ src/
|  ├ app/
|  |  ├ app.component.ts # root component for your app
|  |  ├ app.html # app component template
|  |  ├ app.module.ts # NgModule for app component
|  |  ├ app.scss # global SCSS
|  |  └ main.ts # bootstrap file
|  ├ assets/ # put your images, etc. here
|  ├ pages/ # contains the page components for your app
|  ├ theme/
|  |  └ variables.scss # see https://ionicframework.com/docs/theming
|  └ index.html # main html file
└ www/ # build output directory
```

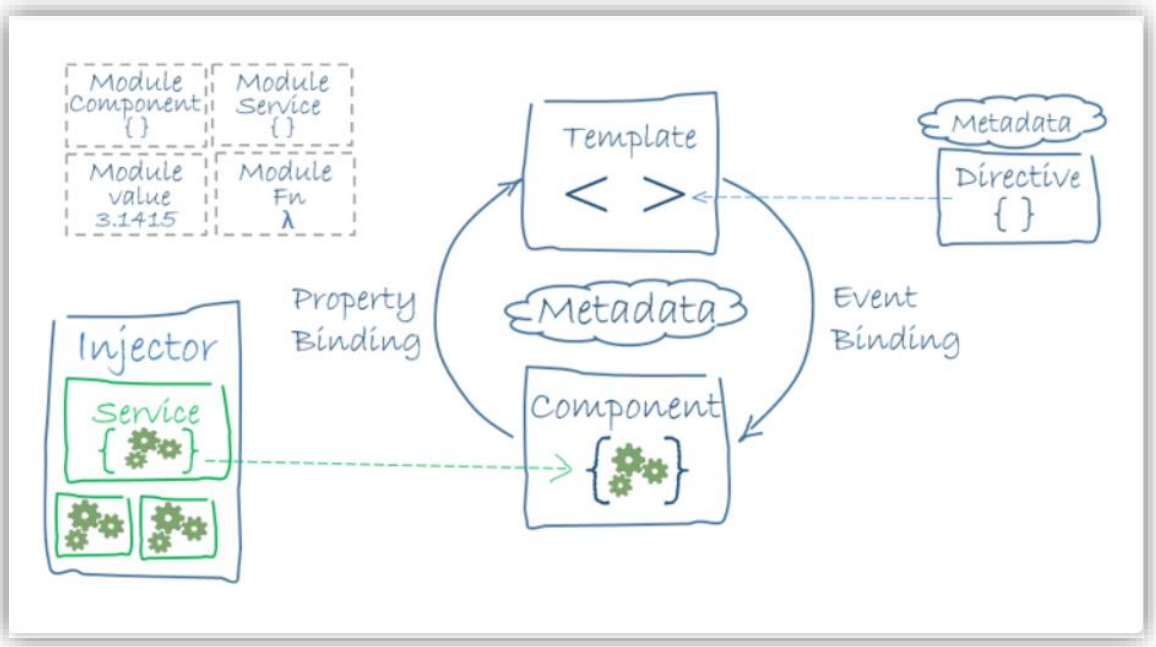
(Drifty, 2017)

L'on peut voir sur cette image que la structure d'un projet Ionic est plutôt simple. L'interface de commande d'Ionic se charge de créer tous les dossiers et fichiers obligatoires pour vous. Certains fichiers permettent de gérer la configuration interne à l'application et ne doivent en aucun cas être modifiés.

Ce qui nous intéresse le plus en tant que développeur, c'est le dossier src. Celui-ci contient tous les fichiers que l'on peut modifier et qui permettent la création de notre application.

Ionic utilise le Framework Angular pour faire la liaison entre le code logique et la vue. Angular offre la possibilité de découper notre code dans un modèle MVC (Modèle, Vue, Controleur) et d'architecturer son projet d'une manière beaucoup plus poussée que lors d'un développement WEB basique. Le fonctionnement architectural d'Angular est décrit dans l'image suivante :

Figure 4 - Architecture d'Angular



(Angular, 2009)

Angular va permettre de créer des composants. Chaque composant représente un objet logique de l'application (une page, une liste, un bouton personnalisé, etc). Chaque composant est relié à un template qui fera office de vue d'affichage pour ce composant. Ce template est écrit en langage WEB basique (HTML / CSS) mais il est possible de rajouter des instructions supplémentaires non existantes dans les langages HTML et CSS. Ces instructions seront comprises uniquement par Angular au moment

de la compilation de notre projet et permettront de faire du binding de données, autrement appelé « Property Binding ».

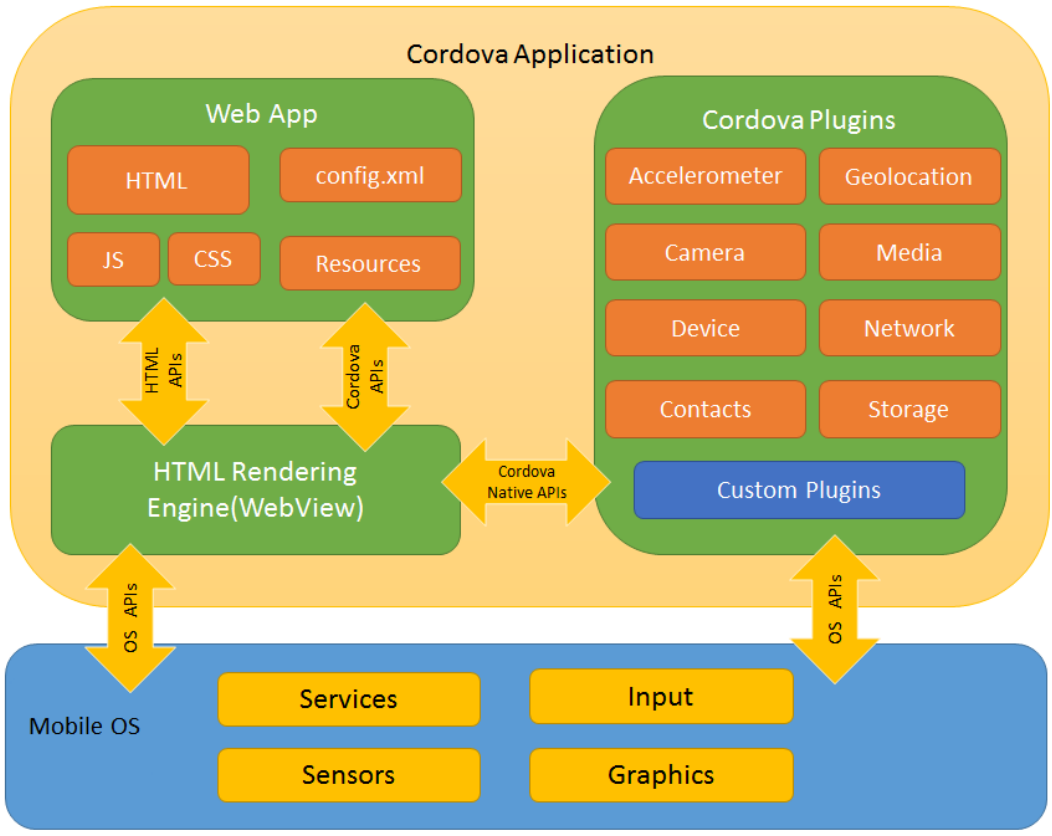
Cela consiste à coller la valeur d'une variable dans la vue de manière dynamique. Si cette valeur change dans la partie logique du composant, Angular va détecter ce changement et automatiquement refaire le rendu du template pour afficher la nouvelle valeur.

Angular permet aussi de faire du « Event Binding ». Cela permet de coller un étiquette à un évènement de la vue du composant en lui disant de lancer une certaine méthode logique lorsque cet évènement est déclenché (ex : lorsque je clique sur ce bouton, lancer cette méthode permettant de changer la couleur du bouton).

Angular utilise aussi le système d'injection de dépendance pour permettre l'accès aux services externes à un composant.

Pour permettre la transpilation du code créer grâce à Ionic, il faut utiliser le Framework Apache Cordova. Celui-ci offre des un accès direct aux composants natif du téléphone. Voici l'architecture de ce dernier :

Figure 5 - Architecture d'Apache Cordova



(Cordova, 2012)

Nous pouvons voir ici qu'Apache Cordova est directement relié au système d'exploitation du téléphone mobile. Une liaison est faite avec le Framework lui permettant d'accéder aux fonctionnalités de bas niveaux du système d'exploitation. A la suite de cela, Cordova propose une palette de plugins permettant d'accéder de manière simplifiée à ces composants de bas niveau.

L'application développée va être compilé dans une WebView. Cette dernier va faire le rendu HTML et afficher les templates que nous avons préalablement créer. La partie logique du code (les composants dans notre cas) vont être compilés dans la Web App. Celle-ci est en constante communication avec la WebView pour y transmettre les données nécessaires.

Apache Cordova offre un système extrêmement simplifiée d'affichage de notre application et d'accès aux composants natifs du téléphone via des plugins.

2.1.2 Avantages

Le plus gros avantage d'Ionic réside dans le fait que c'est un Framework permettant le développement Cross-platform. Le développement Cross-platform est en constante hausse et de plus en plus d'entreprises désirent créer leurs applications en Cross-platform.

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 maintenabilité simplifiée

Le fait que Ionic utilise le langage TypeScript, qui est une surcouche au langage JavaScript simplifie le développement de ce dernier. Le Javascript étant le langage le plus utilisé au monde (Guilloux, 2018), il est très facile de trouver de l'aide et des tutoriels sur internet pour nous aider dans le développement de notre application.

Créer une application sur ce Framework ne nécessite que quelques connaissances basiques en langage WEB (HTML, CSS et JavaScript / TypeScript). Aucunes autres connaissances ne sont nécessaire car tout est tellement simplifé et bien documenté qu'il suffit de regarder dix minutes sur internet pour trouver une solution à nos problèmes.

Un autre grand atout d'Ionic est sa portabilité. Le fait d'avoir du code écrit dans un langage WEB et d'exécuter ce code dans une WebView offre la possibilité d'écrire le même code quelque-soit la système d'exploitation ou tourne l'application. Environ 90% du code écrit peut être compilé sur Android comme sur IOS. Les 10 pourcents restent concerne des plugins d'accès aux composants différents selon les plateformes ou des détails, comme les icons, les images ou du traitement spécifique au cas (taille de l'écran, affichages différents selon l'OS, etc).

Il n'est donc pas nécessaire d'écrire plusieurs fois le même code mais il est nécessaire de l'adapter aux différentes plateformes visées.

2.1.3 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)

2.2 Xamarin

Xamarin est un Framework de développement cross-platform utilisant le langage C#.

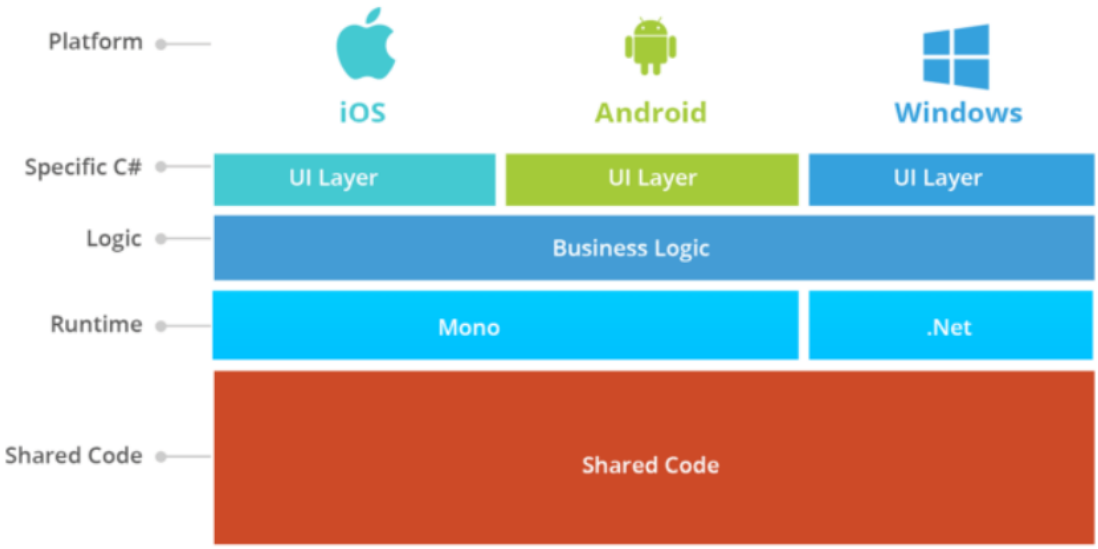
Il permet le développement d'applications Android et IOS en utilisant un seul code logique en C#. La partie visuelle peut, soit être écrite une fois et utilisée pour les 2 plateformes (Xamarin.Forms), soit être écrite pour chaque plateforme. Cette seconde option permet d'être plus proche du rendu que nous voulions car nous n'avons pas de contrainte de type de plateforme. Utiliser la première méthode (Xamarin.Forms) permet d'écrire un seul code fonctionnant sur toutes les plateformes mais nous oblige à faire quelques concessions sur certains designs compliqués.

2.2.1 Architecture

Xamarin dispose d'un ensemble d'outils permettant la compilation de son code sur un téléphone. L'application créée par le développeur ne peut pas directement être compilée sur le téléphone. En effet, les OS actuels ne fonctionnent pas sous C# et ne pourront pas faire tourner une application codée en C#.

Xamarin utilise alors son propre compileur C# et son propre environnement de run-time.

Figure 6 - Architecture de Xamarin



(Nexgendesign, [sans date])

2.2.2 Avantages

Xamarin utilise le langage C#. Celui-ci est langage jeune, en pleine voie de progression. Il est fortement typé, ce qui permet d'y ajouter une couche de protection contre les éventuels bugs générés par un code approximatif.

Les performances que propose une application crée avec Xamarin sont très proches d'une application native. Il faut cependant faire attention du type d'approche que l'on utilise pour créer l'application. En effet, si l'on décide de créer une interface pour chaque plateforme (Xamarin.Android et Xamarin.IOS), les performances seront en effet très proches des performances qu'une application native peut avoir. Par contre si l'on décide d'utiliser le système Xamarin.Forms pour unifier les interfaces des différentes plateformes, les performances pourront en prendre un coup selon le type d'opération faites.

2.2.3 Faiblesses

Le fait que les interfaces soient écrites dans un langage spécifié par Xamarin (XAML, un sort de langage XML) et pas très adaptable nous oblige à créer des interfaces fonctionnelles mais simples. Le design ne peut pas prendre une place prépondérante dans un projet créer avec Xamarin.

Il nous est possible de créer des interfaces unifiées mais le langage utilisé, le XAML, nous rend impossible l'implémentation de designs compliqués. Quand on sait que l'expérience utilisateur et la beauté d'une application sont les 2 points les plus essentiels de nos jours, il est compréhensible de se demander comment Microsoft pourrait améliorer son système pour y ajouter une personnalisation plus poussée.

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)

2.3 React Native

React Native est un Framework se basant sur le Framework React pour créer et composer des interfaces riches à partir de composants déclaratifs codés en JavaScript et JSX.

React Native se distingue des autres Frameworks car il ne compile pas l'application sous forme de « mobile web app » mais sous forme d'application native. Une application native ou une application créée via React Native ne peuvent pas être différenciées au niveau de l'interface. En effet, React Native utilise les mêmes composants de constructions natifs que l'on peut retrouver dans une application native codée en Objective-C ou en Java. React Native se charge juste d'encapsuler ces composants dans des objets compréhensibles et utilisables par le Framework et le développeur. (React Native, 2018)

2.3.1 Architecture

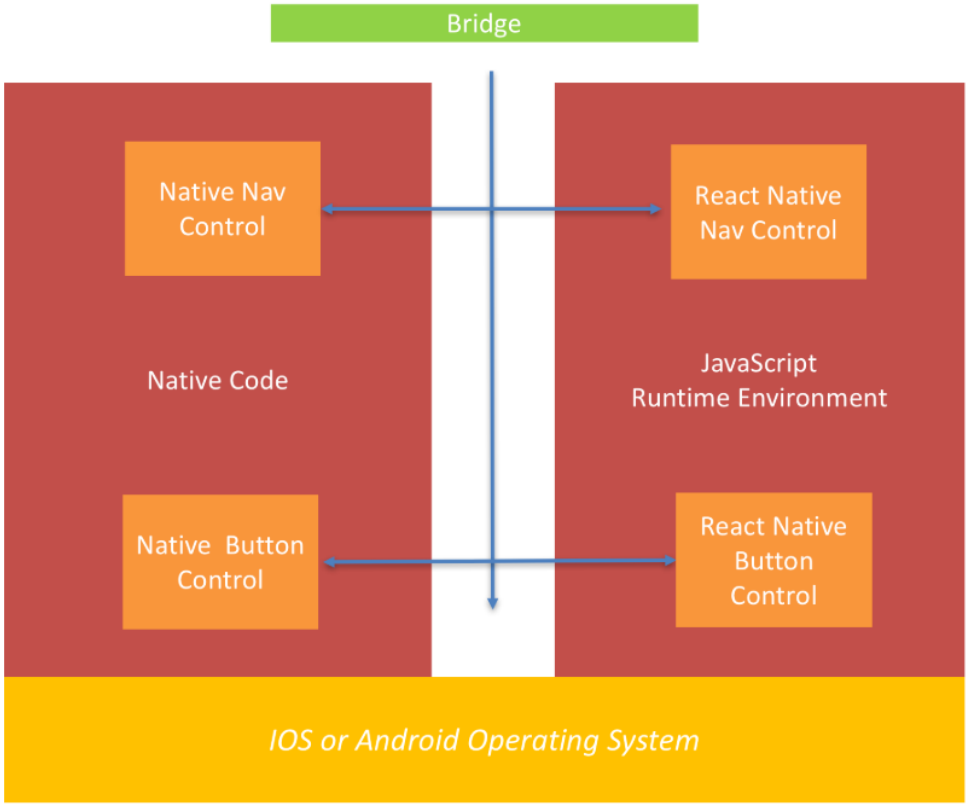
React Native à un fonctionnement très spécifique à lui-même. En effet, un Framework basique tel que Ionic va avoir ses propres composants qui seront connectés via des APIs aux composants natifs du téléphone. Une multitude de couches se superposent alors pour offrir l'accès aux fonctionnalités du téléphone.

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 Natifs.

Il est néanmoins important de comprendre que l'on n'interagit pas immédiatement avec les composants natifs du téléphone. React Natif 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 7 - 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.

2.3.2 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. Environ 90% de l'interface créée pourra être utilisée entre les différentes plateformes et les 10% restant devront être adaptés.

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.

2.3.3 Faiblesses

La plus grande faiblesse de React Native, et sa plus grande force aussi, est son système de liaison aux composants natifs. React Native n'utilise pas le langage Web (HTML/CSS) pour créer ces interfaces vu qu'il n'utilise pas de WebView. La création des interfaces se fait via un langage étendu du JavaScript, le JSX.

Il est donc nécessaire d'apprendre ce nouveau langage (bien qu'il ne soit pas très compliqué).

De plus, il est difficile de designer une application de manière complexe car React Native se réfère directement aux composants natifs qui ont leurs propres design. Certains changements peuvent être fait mais ils sont plutôt basiques.

React Native se base également sur le Framework React pour toute son architecture de composants. Une connaissance du Framework React est alors grandement conseillée.

2.4 Comparaison des Frameworks

Chaque Framework comporte des avantages et des désavantages. Une application créée via le Framework Ionic et Apache Cordova 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 limitée.

2.4.1 Comparaison

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

Tableau 1 – Résumé simple des Frameworks

	Ionic	Xamarin	React Native
Langage	JavaScript / TypeScript et HTML	C# et XAML	JavaScript et JSX
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	Intepréteur
	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)

2.4.2 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)

3. Etude de l’application

3.1 Fonctionnalités de l’application

3.2 Besoins techniques de l’application

4. Choix du Framework

4.1 Concordance avec l’application

4.2 Comparaison avec les autres Frameworks

4.3 Explication détaillée du choix

5. Implémentation de l’application

5.1 Use-case

5.2 Modèle de données

5.3 Choix du système back-end

5.4 Prototypage

6. Développement de l’application

6.1 Apprentissage du Framework

6.2 Environnement de développement

7. Rapport de test

8. Conclusion

Bibliographie

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

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.

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/.

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.

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.

GUILLOUX, Michael, 2018. Quels sont les langages de programmation les plus utilisés par les développeurs ? Une analyse des évènements publics sur GitHub. In : *Developpez.com* [en ligne]. 29 janvier 2018. [Consulté le 18 juin 2018]. Disponible à l'adresse : http://www.developpez.com/actu/185087/Quels-sont-les-langages-de-programmation-les-plus-utilises-par-les-developpeurs-Une-analyse-des-evenements-publics-sur-GitHub/.

KAUDERER, Yan, 2015. Présentation du Framework Ionic | SUPINFO, École Supérieure d'Informatique. In : [en ligne]. 20 août 2015. [Consulté le 18 juin 2018]. Disponible à l'adresse : https://www.supinfo.com/articles/single/155-presentation-framework-ionic.

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

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/.

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/.