

Ecole Pratique des Hautes Etudes Commerciales



Institut d'Enseignement Supérieur Economique de Type Court

**QUEL FRAMEWORK CHOISIR POUR UNE APPLICATION
MOBILE MULTIPLATEFORME DE GESTION DE DETTE?**

Florian GÉRARD

Rapporteur(s)

Travail de Fin d'Études présenté
en vue de l'obtention du diplôme de
BACHELIER EN INFORMATIQUE DE GESTION

Année académique 2018-2019

Remerciements

Durant mon parcours scolaire j'ai eu la chance de croiser plusieurs personnes m'ayant aid  et soutenu, voici l'occasion de les remercier.

Merci au corps enseignant de l'Institut Paul Lambin (o  j'y ai commenc  mes  tudes de d veloppeur) et de la promotion sociale de l'EPHEC pour les cours et les conseils transmis. Un remerciement plus particulier   monsieur Jean-Paul Hecquet, professeur du cours de projet d'int gration, pour m'avoir guid  sur ce TFE en me recadrant sur les points importants.

Petite pens e   toute l' quipe Contraste Consulting, entreprise o  j'ai vraiment pu commencer ma carri re de d veloppeur entour  de personnes bienveillantes et g n reuses en partage de connaissances. Merci   Maxime Debersaques pour tout l'apprentissage et la confiance ainsi qu'  Flavio Occhipinti pour les d bats anim s sur les meilleurs frameworks JavaScript existant et enfin   Nicolas Gielen pour assurer le backend. Je n'oublie pas l'IT Support pour sa bonne humeur et pour avoir fait fonctionner l'imprimante !

Je tenais   ne pas oublier les membres du groupe YAWE (joyeux lurons sachant parfois  tre s rieux) pour les relectures toujours utiles voire m me n cessaires de mes travaux.

Et enfin, merci sp cial   mon meilleur ami Thibault Vanwersch et mon p re Luc G rard pour leurs soutiens inconditionnels au cours de mes  tudes sem  d'emb ches.

Table des matières

Remerciements	1
Table des matières	2
Introduction	5
Money And Relationship eXperience	5
Problématique de la technologie à utiliser pour développer sur support smartphone	5
Description de l'application	7
Schéma de navigation des pages	7
Les écrans	7
Modèle de la base de données	9
Règles	10
L'authentification	11
Les types de relations et currency	12
Routes API du web service	12
Les technologies	21
Base de données	21
Backend	21
Quels frameworks front-end tester et comment?	22
Les choix	22
Ionic	22
Xamarin	22
React Native	23
Autre?	23
Méthode pour éprouver les frameworks	23
Testons les applications hybrides	24
Avant-propos	24
L'environnement de test	24
Le JavaScript	25
Ionic	26
L'initialisation	27
L'affichage et la gestion d'un formulaire	28
La pagination	29
L'interaction avec une API	30
Le multilinguisme	30
	2

Afficher des notifications	31
Essayons d'�tre pr�sentable	31
D�ploiement	32
Autres points importants?	33
Xamarin	34
L'initialisation	34
L'affichage et la gestion d'un formulaire	35
La pagination	37
L'interaction avec une API	38
Le multilinguisme	38
Afficher des notifications	39
Essayons d'�tre pr�sentable	39
D�ploiement	40
Autres points importants?	40
React Native	42
L'initialisation	43
L'affichage et la gestion d'un formulaire	44
La pagination	45
L'interaction avec une API	45
Le multilinguisme	46
Afficher des notifications	47
Essayons d'�tre pr�sentable	47
D�ploiement	47
Autres points importants?	47
Comparaisons r�capitulatif	49
Conclusion	50
D�velopper sp�cifiquement pour mobile est-il toujours utile?	50
D�velopper en pure natif pour Android et iOS n'est pas toujours une perte de temps	50
Sc�nario de recommandation	51
Android et iOS	51
Xamarin	51
Ionic ou React Native	51
Et du coup? Mon choix: Ionic	52
Source	53
Code source du projet	53
Langage - biblioth�ques utilis�es ou �voqu�es	53
Outils utilis�s	53
Autre	53

Introduction

Pour ce travail de développement, je dois m'attaquer à deux problématiques. Premièrement, comment gérer au mieux mes dettes et créances vis à vis de mon entourage.

Deuxièmement, quel est le meilleur framework pour en faire une application disponible sur les appareils de la marque Apple et ceux utilisant Android.

Money And Relationship eXperience

Quand nous avons des interactions sociales, il y a souvent un moment où la question de l'argent arrive : j'ai payé un sandwich à un ami, un collègue m'a prêté 10 euros ou j'ai assumé l'addition d'un restaurant en famille et chacun devra me rembourser sa part.

Et bien évidemment, viendra toujours le moment du remboursement! Mais comment se souvenir de tout ça? La mémoire? On oublie vite ce genre de choses, surtout le montant exact! Une feuille de papier? Elle va vite se perdre!

C'est pour ce type d'occasions qu'une application sur téléphone se révèle utile.

Une fois dans l'application, nous pouvons créer des relations (les personnes qui nous doivent l'argent ou à qui l'on doit de l'argent), et attacher à ces relations des paiements.

Il sera possible d'activer un rappel à un paiement. Celui-ci permettra qu'on reçoive une notification à la date voulue indiquant qu'il serait temps que le remboursement se fasse.

Ce rappel pourrait être mis par défaut en fonction du type de relation de la personne : on pourrait par exemple imaginer qu'on laisse 7 jours avant d'obtenir le rappel pour une relation de type collègue mais qu'on attende 30 jours si c'est pour la famille.

Il me tient également à cœur que cette application soit Open-Source et qu'elle soit donc exploitable par un autre développeur, le plus rapidement et facilement possible! Ce qui fait que j'utiliserais un ORM (Object-relational mapping) afin de ne pas être dépendant d'une base de données particulière.

Problématique de la technologie à utiliser pour développer sur support smartphone

Savoir développer des applications pour plateforme mobile, ce n'est pas se préparer pour l'avenir mais c'est être prêt pour le présent. À présent, et ce depuis 2017, le nombre de personne utilisant un smartphone pour aller sur internet est supérieur que celui sur pc. Passer à côté du développement pour mobile revient à ignorer la majorité des utilisateurs d'appareils numériques.

Mais nous voilà avec un nouveau problème! Les parts de marchés entre les différents acteurs de système d'exploitation sont divisés entre iOS d'Apple (22,85%) et Android de Google (74,45%) (d'après les chiffres de janvier 2019) et ne pas

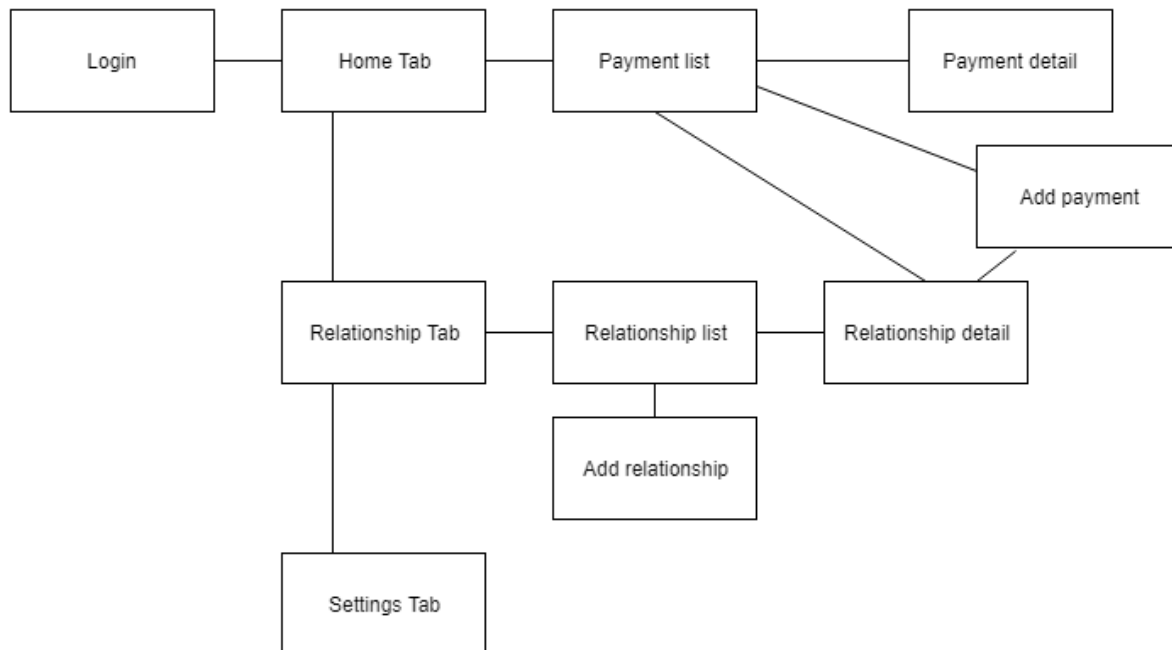
développer pour une de ces deux plateformes c'est se fermer à beaucoup de potentiels clients. Surtout dans le cadre de l'Amérique du Nord où les deux concurrents sont à 49% de part de marché chacun! Malheureusement ces deux OS ont deux langages de programmation natif différents doublant ainsi le travail de développement pour le Front-End. Si nous voulons donc toucher le plus de personne possible avec notre application, nous faudrait-il une équipe avec des informaticiens maîtrisant Objective-C ou Swift pour les appareils à la pomme et Java avec le framework Android pour les smartphones ou tablettes disposant de l'OS Android? Pas spécialement!

La potentielle solution? Le développement d'application Hybride! Le principe, coder dans un seul langage (accompagner d'une bibliothèque ou boîte à outils) qui va être ensuite être traduit en langage natif à la compilation permettant ainsi d'avoir en un seul travail une application mobile tournant sur plusieurs types d'appareil.

Mais des outils pour faire cela, il y en a énormément, mon but ici est de vous présenter les plus pertinents et de vous conseiller sur lequel serait le meilleur (le tout était souvent très subjectif restons honnête).

Description de l'application

Sch ma de navigation des pages



Les  crans

Tabulation:

Une fois la connexion faite, nous sommes redirig  sur la page d'accueil et avons comme menu une tabulation en bas vers la page d'accueil (celle de base), des options et la page des relations.

 cran d'accueil:

La liste des 5 paiements non rembours s les plus vieux.

Chaque  l ment de la liste est cliquable pour acc der   ses d tails.

Un bouton "ajouter" sera mit en bas pour ajouter un nouveau paiement.

route utilis e: [getPayment](#)

 cran des options:

Dans les options, l'utilisateur pourra changer la langue de l'application (l'anglais et le fran ais seront propos s de base mais d'autre langues seront proposables facilement par la suite).

Cela sera  galement ici que nous pourrons ajouter des types de relations.

 cran des relations:

La liste des relations de l'utilisateur.

Chaque  l ment de la liste affiche le nom et le type de relation. On peut aussi acc der aux d tails de chaque relation via cet  cran.

On pourra directement ajouter une relation sur cet  cran.

Un clic sur une relation permettra d'aller dans ses d tails.

routes utilis es: [getAllRelation](#) , [addRelation](#)

D tails d'une relation:

Y sera affich  de mani re  ditable le nom de la relation et son type.

Une liste d'historique des paiements sera pr sente avec en premier ceux non rembours .

Chaque  l ment de la liste sera cliquable pour en voir les d tails.

Un bouton "ajouter" sera mis en bas pour cr er un nouveau paiement li    cette relation.

On pourra  galement depuis ici supprimer la relation

Routes utilis es: [getRelation](#) , [updateRelation](#) , [deleteRelation](#)

D tails d'un paiement:

Dans les d tails seront pr sentes les informations du nom de la relation, le titre du paiement ainsi que le d tail si pr sent, le montant avec la devise, la date de cr ation ainsi que la date du remboursement si il a  t  rembours .

Un bouton "rembours " sera pr sent pour indiquer que le paiement est rembours .

S'il a d j   t  rembours , une ic ne verte sera pr sente.

On pourra  galement via cet  cran ajouter un rappel.

Routes utilis es: [getPayment](#) , [refundedPayment](#) , [getReminder](#) , [addReminder](#)

Nouveau paiement:

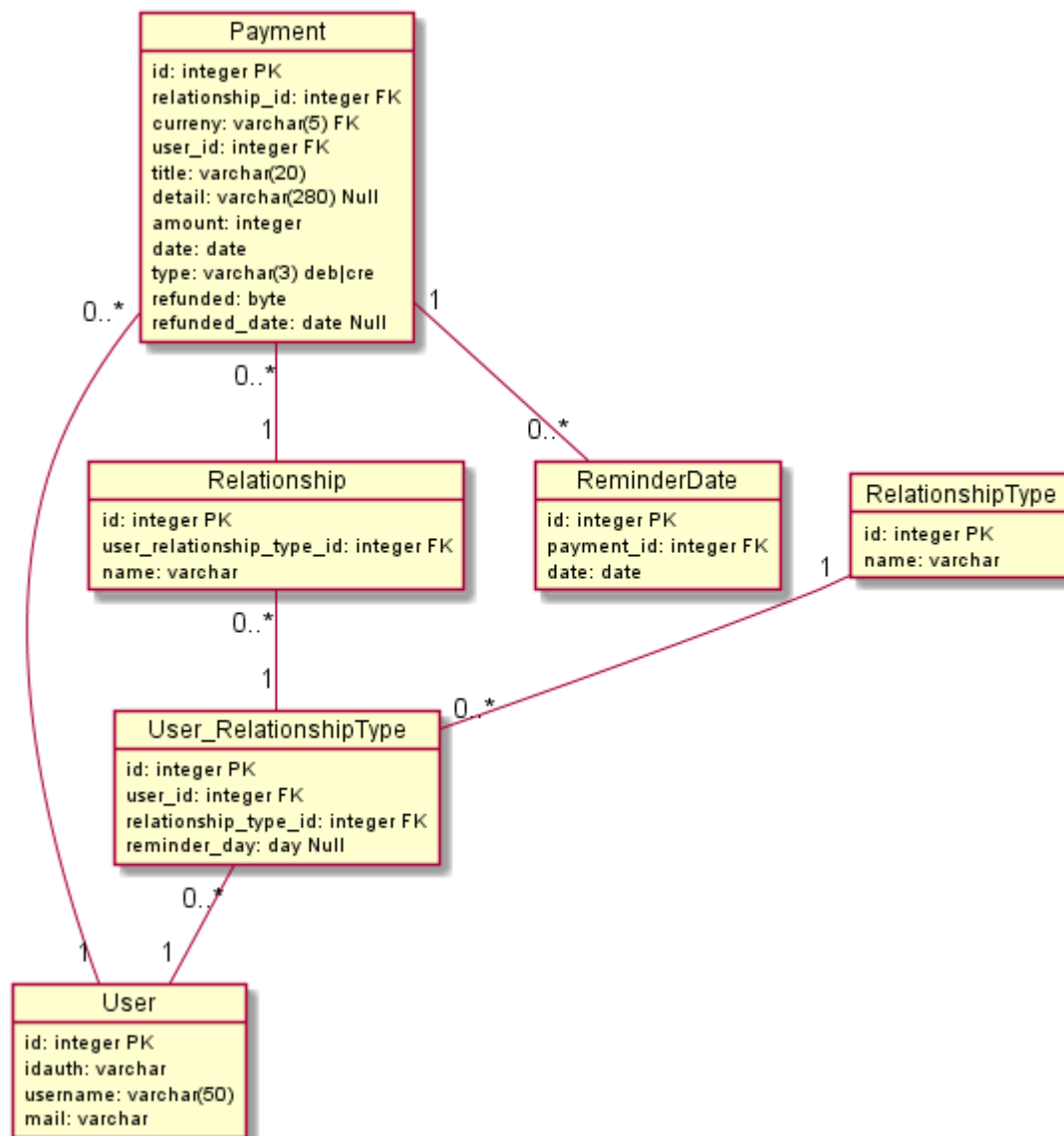
Cet  cran peut  tre appel  de diff rents endroits : depuis l' cran d'accueil, depuis l' cran r capitulatif des paiements et depuis le d tail d'une relation.

Sur cet  cran sera disponible un formulaire afin d'enregistrer un nouveau paiement.

Si nous venons de la page de d tail d'une relation, le champ relation du formulaire sera automatiquement rempli avec la cat gorie li e   cette relation.

Routes utilis es: [addPayment](#) , [getAllCurrencies](#) , [getAllRelationship](#)

Mod le de la base de donn es

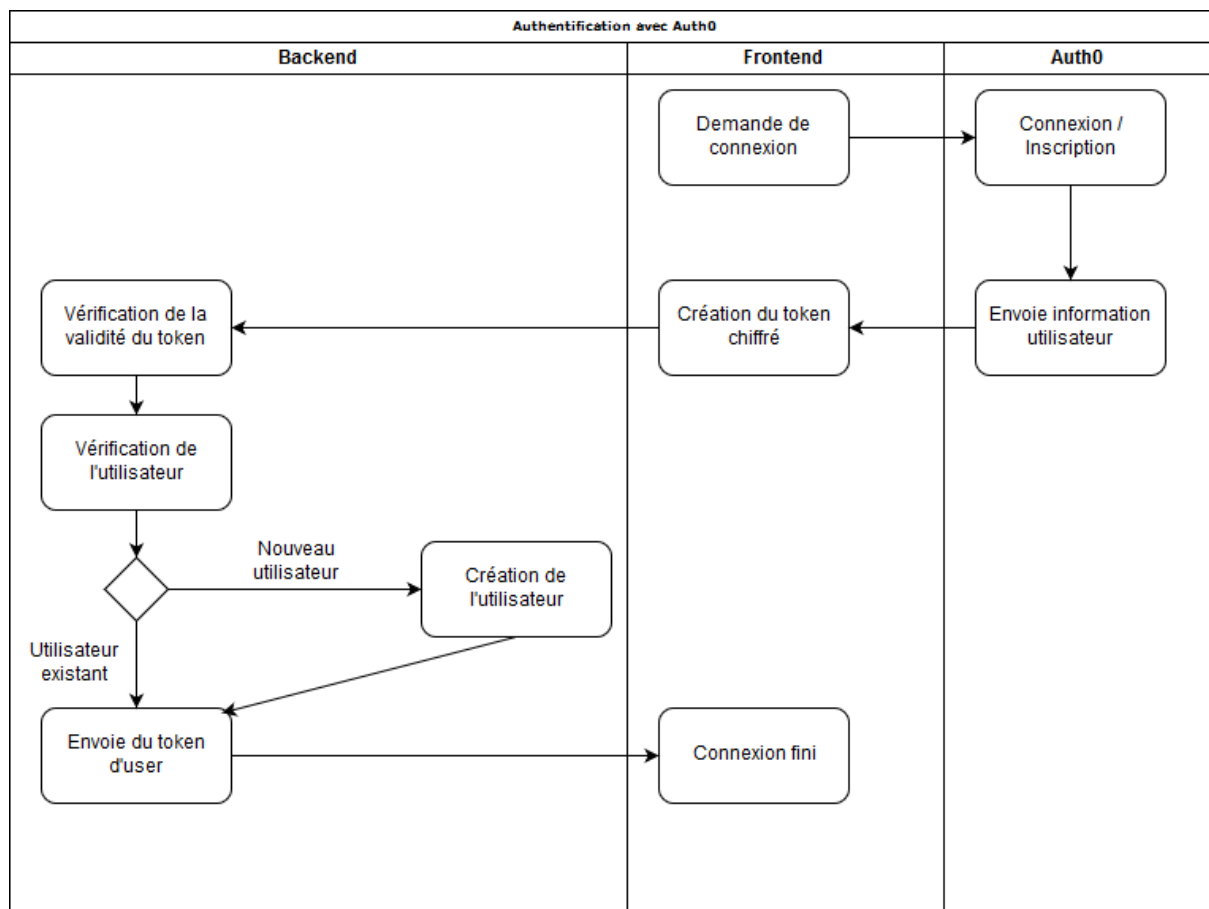


R gles

- Un utilisateur doit poss der un nom, une adresse mail et un ID venant d'Auth0. Il peut avoir plusieurs types de relations ainsi que plusieurs devises. L'ID venant d'Auth0 est unique.
- Un type de relations doit poss der un nom et peut appartenir   plusieurs utilisateurs. Le nom est unique.
- Dans la table de liaison entre un utilisateur et un type de relations, l'ID de l'utilisateur et l'ID du type forment une cl  unique.
- Une relation doit avoir un nom ainsi qu' tre li e   un unique utilisateur et   un unique type de relations.
- Une devise doit poss der un nom de maximum 50 caract res et un label de 3 caract res. Elle peut appartenir   plusieurs utilisateurs. Le nom est unique.

- Un paiement doit poss der une unique devise et appartient   une relation unique, il doit poss der un montant, une date, un type qui sera soit "det" pour dette soit "cre" pour cr ance ainsi qu'une date de remboursement.
- Un rappel de paiement doit appartenir   un paiement et doit poss der une date.
- Un rappel de paiement par type de relations doit appartenir   un type de relation et doit poss der un nombre de jours.
- Un type de relations ne peut pas  tre supprim  si une relation la poss de.

L'authentification



Pour s'authentifier   l'application, on passera via le service Auth0 (<https://auth0.com/>), ce qui permettra d'utiliser au choix de l'utilisateur son compte Google, Facebook ou Hotmail. L'authentification classique avec adresse e-mail et mot de passe sera  galement possible.

Auth0 me renvoie un token contenant comme information un ID d'utilisateur que je traite ensuite. Je regarde dans ma table "user" si une personne avec cette ID d'Auth0 existe. Si c'est le cas il est redirig  vers la page d'accueil. Si le web service le rep re en tant que nouvel ID, on lui demandera plus d'informations pour finaliser son inscription.

Passer via le service Auth0 permet   l'utilisateur de passer par un de ses comptes pour se connecter (m thode de plus en plus populaire). De plus, la gestion du mot de passe est faite par un service reconnu et permet  galement d'avoir un rappel par

mail de ce genre d'informations (ne poss dant pas de serveur mail, j'en avais besoin).

Pour la s curit  du backend, un token chiffr  reprenant les informations d'Auth0 sera envoy  depuis le frontend, afin de les comparer avec le Json re u afin d' viter les faux comptes ne passant pas via mon compte Auth0.

Les types de relations et currency

Un user peut rajouter un type de relation pour ses relations. Quand celui-ci le fait, le web service v rifie d'abord que ce type existe. Si c'est le cas, on rajoute dans la base de donn e une liaison dans User_RelationshipType, sinon on ajoute d'abord ce nouveau type de relations dans RelationshipType et ensuite on fait la liaison. Cela  vite d'avoir des doublons dans RelationshipType.

Gr ce   cela l'user peut se voir proposer des types de relations quand il en veut un nouveau, sans pour autant avoir la liste de toutes les RelationshipType de tout le monde. Cela fait aussi un gain de m moire pour la base de donn es.

Le principe sera le m me pour les Currency.

Routes API du web service

Voici les descriptions de toutes les routes du web service. Cette documentation servira   ce que n'importe quel projet Front-End puisse l'utiliser facilement.

G n ralit s : toutes les routes (except  /login) doivent avoir dans le header la propri t  "Api-Token" qui sera le token re u par la route /login. Ceci me permettant de s curiser les routes, mais  galement de faire transiter automatiquement des informations comme l'ID de l'user (utile lors de nombreuse requ te).

Les requ tes et les r ponses seront envoy es en Json.

Si il y a une erreur dans le backend, celui-ci renverra un Json avec la propri t  message qui est un tableau de string d crivant les erreurs (car il pourrait y avoir plusieurs soucis   afficher).

login

Les d tails de fonctionnement de cette route se trouvent dans le chapitre ["L'authentification"](#)

POST	login
Request	<pre>{ user_id varchar name varchar }</pre> <p>l'user_id �tant le token renvoy� par Auth0 un token sera envoy� avec, mais pas le m�me que les autres routes</p>

Response	{ api-token varchar }
----------	-----------------------------

updateName

PUT	user
Request	{ name varchar }
Response	{ name varchar }

allRelationshipType

GET	relationshipType
Request	
Response	liste de tous les types de relations existantes dans la base de donn�es { id integer name varchar }

allRelationshipTypeForUser

GET	user/relationshipType
Request	
Response	liste des types de relations li�es � l'utilisateur courant { id integer relationship_type object { id integer name integer } }

addRelationshipTypeForUser

POST	user/relationshipType
Request	{ name varchar }
Response	{ id integer name varchar }

deleteRelationshipTypeForUser

DELETE	user/relationshipType/{id}
Request	
Response	

getAllRelationship

GET	user/relationship query param�tre: name, relationship_type
Request	possibilit� de trier les relations par type de relations ou de faire une recherche sur le nom
Response	liste des relations de l'utilisateur courant d�tail de la relation { id integer name varchar user_relationship_type object { relationship_type object { name varchar } } }

getRelationship

GET	user/relationship/{id}
Request	
Response	d�tail de la relation <pre>{ id integer name varchar user_relationship_type object { relationship_type object { name varchar } } }</pre>

addRelationship

POST	user/relationship
Request	<pre>{ name varchar relationship_type_id integer }</pre>
Response	la nouvelle relation <pre>{ id integer name varchar user_relationship_type object { relationship_type object { name varchar } } }</pre>

updateRelationship

PUT	user/relationship/{id}
Request	<pre>{ name varchar }</pre>

	<pre>relationship_type_id integer }</pre>
Response	<pre>la relation mise � jour { id integer name varchar user_relationship_type object { relationship_type object { name varchar } } }</pre>

deleteRelationship

DELETE	user/relationship/{id}
Request	
Response	<pre>la relation supprim�e { id integer name varchar relationship_type object { name varchar } }</pre>

allCurrency

GET	currencies
Request	
Response	<pre>Liste de toutes les currencies { id integer name varchar label varchar }</pre>

allCurrencyByUser

GET	user/currencies
Request	
Response	Liste de toutes les currencies utilis�es par l'utilisateur courant { id integer user_id integer currencies_id integer currency object { id integer name varchar label varchar } }

addCurrency

POST	user/currencies
Request	{ name varchar label varchar }
Response	le currency nouvellement ajout�

getAllPayment

GET	user/payment query param�tre: type, refunded, relationship_id, number_row
Request	query optionnel pour filtrer le type (deb ou cre) ou si c'est d�j� rembours� ou non (par d�faut, prend tout) ainsi que le relationship_id si on veut tous les paiements li�s � un utilisateur number_row permet de s�lectionner combien on veut en afficher
Response	liste des paiements de l'user courant : { id integer amount integer title varchar description varchar }

	<pre> date date type varchar refunded byte refunded_date date relationship object { id varchar name varchar } user_currency object { id integer currency object { id integer name varchar label varchar } } </pre>
--	--

getPayment

GET	user/payment/{id}
Request	
Response	<p>un paiement retrouv� via son ID :</p> <pre> { id integer amount integer title varchar description varchar date date type varchar refunded byte refunded_date date relationship object { id varchar name varchar } user_currency object { id integer currency object { id integer name varchar label varchar } } } </pre>

addPayment

POST	user/payment
Request	{ amount integer title varchar description varchar (optionnel) type varchar (deb cre) date date user_currency_id integer relationship_id integer user_id integer }
Response	le paiement nouvellement cr�� voir getPaymentDetail

refundedPayment

PUT	user/payment/refunded/{id}
Request	Aucune information n'est envoy��e en plus de l'ID du paiement qui est � pr��sent rembours�� La date du remboursement se fait dans le backend { }
Response	le paiement � pr��sent rembours�� voir getPaymentDetail

updatePayment

PUT	user/payment/{id}
Request	{ title varchar description varchar (optionnel) user_currency_id integer relationship_id integer amount integer }

Response	le paiement mis � jour voir getPaymentDetail
----------	---

deletePayment

DELETE	user/payment/{id}
Request	
Response	

getReminderDate

GET	user/reminderdate
Request	
Response	Toutes les dates de rappel de paiement (avec le paiement) de ceux qui sont non-rembours�s { id integer date date payment object { voir getPayment } }

postReminderDate

POST	user/reminderdate
Request	{ paymentId integer date date }
Response	Le reminder date nouvellement ajout� voir getReminderDate

deleteReminderDate

DELETE	user/reminderdate/{id}
--------	------------------------

Les technologies

Base de donn es

Ma base de donn es sera une relationnel MySQL. Je dispose d j  d'un h bergement web chez OVH, je l'exploite donc pour  viter des surco ts. De plus, MySQL r pond parfaitement   mes besoins de stocker de donn es, pas besoin de plus.

Je dois juste faire attention car je ne poss de qu'une seule base de donn es, partageant ainsi tous mes projets. Pour ne pas m'y perdre, je nomme chacune de mes tables avec l'abr viation du projet. Ici,  a sera "marx", ainsi par exemple, la table des users s'appellera "marx_user".

Backend

Toujours dans l'optique d' viter de payer pour rien, j'utilise mon serveur OVH pour y faire tourner un webservice. Dans mon abonnement je ne peux faire tourner que du PHP, c'est donc dans cette famille de langage que j'ai cherch  un framework pour faire une API.

Mes besoins sont simples, pouvoir interagir avec ma base de donn es le plus facilement possible (l'utilisation d'un ORM sera donc souhait ) et communiquer avec des JSON.

Il existe deux grands frameworks reconnu en PHP, Symfony et Laravel.

Les deux sont assez lourd, ils sont fait pour  galement afficher des sites webs (ce dont je n'ai pas du tout besoin).

Heureusement, l' quipe de Laravel   sorti une variante plus l g re, utile pour uniquement faire des API. Parfait, c'est ce dont j'ai besoin.

Lumen est son nom. Et en ce qui concerne l'ORM, l' quipe de Laravel vient   nouveau   mon secours en ayant cr   Eloquent.

Quels frameworks front-end tester et comment?

Les choix

Des bo tes   outils pour faire des applications hybride, il y en a   la pelle, je dirais m me qu'il y en a de trop. L'impression que chaque d veloppeur veut monter le sien pour la reconnaissance se fait vite ressentir et on se retrouve avec beaucoup de fork de projet open-source dans l'univers JavaScript. J'ai donc d  faire un choix en fonction de ceux apportant une r elle plus-value dans ce monde fort morcel  du d veloppement mobile hybride et j'en ai tir  trois frameworks populaires. Parmi mes choix, deux frameworks JavaScript vont ressortir car except  Xamarin de Microsoft il n'existe que cela pour faire des applications multiplateformes.

Ionic



Pionnier en la mati re, je ne pouvais donc pas passer   c t  de celui qui a popularis  le d veloppement multiplateforme en JavaScript.

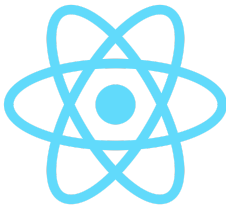
Bas  sur Angular (qui est un coup de c ur personnelle du Front-End), Ionic   37 000 fans sur Github, fork  12 000 fois et ayant 293 contributeurs (chiffre du 26 f vrier 2019). Ionic a sorti sa premi re version en 2013.

Xamarin

Derri re Xamarin se cache une histoire un peu complexe. De base Xamarin existait afin de faire tourner des applications .Net de Microsoft sur les distributions Unix (GNU/Linux et MacOS) et s'est transform  petit   petit en framework C# afin de faire de l'Android et de l'iOS en un seul code avec les outils Microsoft. Celui-ci a d'ailleurs rachet  le projet (et le personnel derri re) en 2016 pour y apporter son soutien financier et logistique.



React Native



Soutenue par Facebook, la premi re version de React Native est sortie mars 2015 et a su vite se faire appr cier gr ce aux fans de base de React JS sur lequel ce framework en JavaScript se base. 74 000 fans sur Github et avec 1800 contributeurs le 26 f vrier 2019, React Native est le mastodonte du d veloppement iOS / Android actuellement et reste un incontournable   tester.

Autre?

En testant les trois frameworks cit s pr c demment, j'explore essentiellement le JavaScript avec Angular et React (qui sont les 2 frameworks du moment pour faire du Front-End) et je n'oublie pas de jeter un  il   ce que propose Microsoft (qui reste un des plus gros acteurs de l'informatique) dans ce secteur.

Mais il existe bien plus de biblioth que qu'on pourrait tester.

PhoneGap lanc  par Adobe reposant aussi sur le JavaScript, Onsen UI, Framework Seven, Quasar  galement en JavaScript sans oublier NativeScript qui monte de plus en plus en force et qui nous permet de faire du mobile en Vue JS (concurrent   Angular et React).

M thode pour  prouver les frameworks

Pour savoir   quel point ces frameworks sont agr ables   utiliser, je testerais plusieurs  l ments importants   mes yeux.

- L'initialisons d'un projet et la clart  de la documentation
- La gestion de l'affichage, comment montrer les donn es, interagir avec eux et utiliser un formulaire
- La pagination, comment est g r  le passage d'une page   l'autre
- L'interaction avec le backend, y r cup rer des informations et en envoyer
- Savoir y g rer plusieurs langues
- Afficher des notifications
- Avoir un rendu graphique le plus beau possible
- Comment d ployer l'application sur smartphone

Je me baserais essentiellement sur la documentation officielle de chaque framework pour d velopper l'application car selon moi un framework efficace est aussi celui qui a la meilleure documentation possible, c'est un crit re essentiel   mes yeux.

Testons les applications hybrides

Avant-propos

L'environnement de test

Afin d'éprouver nos applications que nous développons, nous bénéficions d'émulateur de smartphone. Les plus reconnus sont fournis par l'ADM (Android Device Manager) de Google disponible via l'Android Studio (IDE de Google en collaboration avec JetBrains) mais également via Visual Studio.

Nous pouvons également tester directement sur notre propre appareil Android si il est relié au pc. Ceci est plus performant (un émulateur est très lourd en ressource de mémoire vive) mais nous permet pas de tester sur plusieurs types d'écrans.

Précisions importante, je n'ai pas testé les frameworks dans leurs versions iOS. Certes je fais un travail de veille technologique pour applications multiplateformes et cela serait donc requis que je teste le tout sur plusieurs plateformes, cependant pour pouvoir tourner mon code sur les appareils Apple, je dois posséder justement un Mac (pour la compilation et également l'émulation) ce que je ne détiens pas.

Pour les déploiements, je ne ferais donc cela qu'en Android. Je serais à même de fournir un APK (Android Package, un format de fichier pour exécuter des installations sur appareil Android, équivalent du .EXE de Windows) mais je ne pourrais pas déployer cela sur le Google Play Store (application où la grande majorité des personnes téléchargent leurs logiciels) car cela est payant (25 dollars à vie) et que je ne possède pas de carte de crédit pour ça. Précisons tout de même que pour le market d'iOS, il faut déboursier 99 dollars par an.

Le soucis en ne passant pas via un market d'application est que l'utilisateur va devoir dans les options de son appareil autorisé manuellement les applications externes ce qui peut l'effrayer (un message d'avertissement ne rassurant jamais apparaît).

Vue que je testerais sur du un émulateur Android, j'installerais Android Studio me permettant d'installer automatiquement et un émulateur mais également les outils Java qui compileront mon code, c'est à dire le JDK (Java Development Kit) et l'Android SDK (Software development kit) sans quoi je ne saurais rien produire.

En ce qui concerne les outils de développement, pour Xamarin j'utiliserais l'IDE Visual Studio 2017 qui est fourni par Microsoft et j'exploiterai Visual Code qui est un excellent éditeur de texte se transformant aisément en IDE en y ajoutant quelques plugins.

Le JavaScript

Au vue de la multiplicit  des frameworks JavaScript, je trouvais important de vous pr senter avant ce langage et son  volution.

1996, ce langage sort avec le navigateur Netscape 2 (actuelle Firefox) et il apporte un gros changement au d veloppement de page web, l'interactivit  avec l'utilisateur sans devoir recharger la page. Cela peut  tre en g rant des formulaires plus efficacement en indiquant des messages d'erreurs directement ou en int grant des animations plus fantaisiste.

Pour ce faire, le code JavaScript nous permet d'interagir avec le DOM (Document Object Model) en y modifiant l'HTML (langage balis  formant nos page web) et chang    la vol  celui-ci.

Ainsi, ce langage est vu comme une aubaine et permet d'avoir des sites dynamiques sans devoir tout le temps rafra chir la page.

Mais comme souvent dans le monde informatique, tout n'est pas rose! Le JavaScript est un langage interpr t , cela veut dire que chaque navigateur va le lire comme il l'entend ce qui va poser des soucis de compatibilit  de code et arrachera les cheveux de nombreux d veloppeur qui vont devoir travailler   ce que leurs sites fonctionnent partout. ECMAScript va appara tre en 1997 pour normaliser ce langage aupr s des distributeurs de navigateur. Sans r el grand succ s.

Une solution est apparue avec des biblioth ques traduisant le JavaScript normalis  en fonction du navigateur o  est interpr t  notre code (cela s'appelle le Polyfill). Le plus populaire fut le JQuery lanc  en 2006. Bien  videmment cela rend les programmes moins performant mais fort heureusement les navigateurs ont de plus en plus tendance   respecter l'ECMAScript.

Ce langage a  galement la caract ristique de ne pas  tre typ . On ne d finit pas   l'avance si une variable sera un string, un chiffre, un boolean ou tout autre type pouvant exister.

Le JavaScript   prit un tournant majeur en 2009 quand NodeJS est pr sent  par son cr ateur Ryan Dahl. NodeJS permet tout simplement de pouvoir ex cuter du JS en dehors du navigateur! Pour ce faire il se base justement sur le moteur JavaScript de Google, V8.

Cela permet par exemple de faire du rendu serveur en JavaScript (du backend par exemple) et a surtout permis l' mergence de framework tel Angular dont on en parlera plus tard avec Ionic mais  galement d'outil de testing comme Karma ou Jasmine.

Autre avanc  non n gligeable, l'apparition de NPM (Node Package Manager) qui permet le partage en une ligne de commande de biblioth ques ( quivalent de NuGet pour C# ou Composer pour PHP). C'est devenu un  l ment indispensable en JavaScript.

Ionic

Ce framework Cross-Plateforme   sorti sa premi re version stable en 2013 en se voulant respectueux des standards du web mais pour faire du mobile.

Avant de parler   proprement dit d'Ionic, je vais devoir introduire Angular et Cordova car il les utilise tous les deux afin de permettre aux d veloppeurs de r aliser des projets d'application mobile.

Commen ons avec Apache Cordova qui est le moins le connu.

Celui-ci permet d'utiliser des fonctionnalit s natives du t l phone (la cam ra, le gps, afficher des notifications, ...) et est compatible avec de nombreux syst me d'exploitation mobile (Android et iOS bien  videmment mais  galement Windows Phone, Blackberry et Symbian l'ancien OS de Nokia). Cordova utilise les technologies web pour fonctionner.

En ce qui concerne Angular, framework d velopp  par Google depuis 2009, il apporte essentiellement une structure, une architecture aux applications webs o  chaque composant (on ne parle plus de page) aura son propre HTML pour la forme, son JavaScript qui sert de contr leur et son style CSS (avec toujours la possibilit  d'avoir un CSS global mais c'est n'est pas recommand ).

Cet outil de Google utilise depuis 2016 TypeScript qui est une surcouche de JavaScript permettant de typer les variables (les d finir comme cha ne de caract re, nombre, ...) et ainsi permettre une meilleur lisibilit  du code.

Angular utilise des directives pour structurer l'HTML. Ceci permet de changer dynamiquement le DOM via certains attribut.

Ainsi nous pouvons directement mettre des conditions (if) ou des boucles (for) dans notre fichier HTML

```
<ion-spinner *ngIf="isLoading" item-start name="dots"></ion-spinner>
```

```
<ion-list *ngIf="list && !isLoading">
  <button
    ion-item
    *ngFor="let item of list"
    (press)="delete(item)"
  >
    {{ item.relationship_type.name }}
  </button>
</ion-list>
```

Dans cet exemple, la directive *ngIf permet d'afficher une animation d'attente si la variable (d finie dans le fichier JavaScript) est   True

Nous pouvons  galement mettre plusieurs conditions facilement comme pour la liste qui ne sera affich e uniquement si `isLoading` est `False` et que la `"list"` (variable toujours g r e en JavaScript) est  galement pr sente.

Le `*ngFor` lui servira   cr er plusieurs bouton (un par  l ment de la liste) et chaque donn e de la liste prendra le nom de variable `item` comme d sign  ainsi. Cela servira pour afficher la propri t  `relationship_type.name`   chaque occurrence de bouton et surtout lui lier comme il faut le bon objet   supprimer.

Pour lier une m thode   un  v nement (un clic, une longue pression, ...) il suffit d'indiquer   quoi nous voulons r agir (ici une longue pression sur l' l ment bouton) et lier   la m thode JavaScript (`delete()` dans le cas pr sent) avec, s'il le faut des param tres.

Autre grand point important d'Angular, le two-way data-binding. C'est le fait de refl ter automatiquement les changements de variables entre le contr leur (en Typescript) et la vue (en HTML). Principe tr s puissant r duisant le c  t de d veloppement car nous ne devons plus manuellement changer la valeur d'une variable dans l'affichage si elle est chang e dans le javascript et vice versa.

Parlons  galement de l'injection de d pendance, un design pattern o  nous cr ons des singletons (aussi appel  service) envoyant des donn es aux diff rents composant. Cela  vite la duplicit  du code et en isolant les services (utiliss s pour appeler l'API) les tests automatiques sont simplifi s.

Ionic allie ainsi la puissance de Cordova pour interagir avec les plateformes mobiles et la structure de d veloppement d'Angular afin d'avoir une architecture coh rente limitant la r p tition de code et permettant  galement aux habitu s du web avec Angular de vite retrouver leurs pratiques.

Teston d s   pr sent les joies du d veloppement avec Ionic

L'initialisation

Pour commencer un projet Ionic, nous devons passer par la ligne de commande. En installant via NPM Ionic de mani re globale ses outils, une flopp e de commandes nous sont fournis (on appelle cela la CLI). La commande `"ionic start"` nous fait appara tre un questionnaire demandant le nom du projet et nous demande ensuite de s lectionner un template pour commencer. Les templates nous permet d'avoir d j  un squelette de navigation en nous proposant! un menu apparaissant sur demande sur le c  t  ou le menu en bas de page avec des ic  nes. Ionic nous propose  galement le template `"super"` qui fournit de nombreux exemple et bonne pratique pour le d veloppement, tr s utile quand on d bute et que nous sommes perdu.

En ce qui concerne l'architecture fourni de base, nous irons vite l'am liorer car au d but Ionic ne fait pas la diff rence entre les pages et les simples composants. Mais lorsque nous g n rons automatiquement ceux-ci (merci au CLI) un dossier pages et composant appara tront automatiquement (alors autant le faire directement).

La documentation sur le site d'Ionic est  galement bien fournie avec   chaque fois des exemples de codes facilement visible et le template `"super"` lors d'une cr ation de projet aide   avoir une belle vue des possibilit s.

L'affichage et la gestion d'un formulaire

Utilisant Angular, nous d velopperons le style via le CSS, l'interaction en TypeScript et la forme dans un document HTML mais en employant des balises propre   Ionic. Nous pouvons toujours nous servir de balise propre   HTML mais cela est fortement d conseill .

Chaque composant Ionic commence avec ion dans le nom et permet d'avoir directement des  l ments que les utilisateurs mobiles sont habitu s   voir. Citons par exemple ion-fab, qui permet d'avoir un bouton rond avec une ic ne restant toujours au m me endroit de la page (tr s souvent en bas   droite) ou ion-datetime qui fait appara tre un s lectionneur de date que l'on peut d rouler pour choisir le jour voulu.

En ce qui concerne la structure des pages, Ionic cherche toujours   se rapprocher du web et propose ainsi qu'on puisse mettre un header et un footer si besoin. Le component ion-content est l  pour y mettre le contenu de la page. Pour g rer la position des  l ments, Ionic recommande grandement d'utiliser le syst me des grilles. Ceci a  t  popularis  par Bootstrap, biblioth que CSS, et nous permet de voir l'affichage comme un damier de jeu d' chec o  nous pouvons ainsi positionner et d limiter les  l ments bien plus facilement.

Comme vue pr c demment, nous avons des directives que nous pouvons placer directement dans les balises HTML de nos composants.

Cela permet de facilement afficher ou cacher des  l ment de la page sous certaines conditions.

Explorons maintenant la gestion d'un formulaire. Angular nous fournit le service FormBuilder nous permettant via une variable de d cr ter les noms de champs qu'on a besoin et d'y ajouter une validation

```
this.relationshipForm = this.fb.group({  
  name: ['', Validators.required],  
  relationshipTypeId: ['', Validators.required]  
});
```

Dans l'exemple nous avons un formulaire pour une relation qui va avoir un nom (requis) et un id de type de relation ( galement requis).

```
<form [formGroup]="relationshipForm">
```

Dans l'HTML, via la balise form (non sp cifique   Ionic ou Angular) on lui lie la variable.

```
<ion-select  
  interface="popover"  
  formControlName="relationshipTypeId"  
>  
  <ion-option  
    *ngFor="let item of relationshipTypeList"  
    [value]="item.id"  
    >{{ item.relationship_type.name }}</ion-option  
  >  
</ion-select>
```

Nous lions ensuite chaque  l ment de la variable de formulaire   des composants Ionic permettant de former un formulaire. Ici une liste d' l ment   choisir. La directive `fromControlName` permet de savoir   quoi lier l' l ment dans le formulaire. Et cela se fait automatiquement gr ce au two way data-binding! Je n'ai pas besoin de rajouter un  v nement qui   chaque s lection change la variable ou   chaque entr e clavier renouvelle le contenu de cette variable dans le cadre d'un champ de texte.

```
<button
  type="submit"
  color="secondary"
  [disabled]="!relationshipForm.valid"
>
```

Une autre grande puissance des formulaires en Angular, c'est via sa validation, ma variable poss de plusieurs propri t  afin de v rifier des conditions, comme ici je peux rendre un bouton indisponible si le formulaire n'est pas valable. Tr s pratique pour emp cher d'actionner des  v nements (comme un ajout dans la base de donn es) si les conditions ne sont pas requises et pour  galement indiquer des messages d'erreurs ou de pr ventions.

La pagination

Ionic propose deux mani re de g rer la pagination des pages.

La premi re n'est nul autre que la gestion   la Angular, le routing en y important le module Router. Pour aider   comprendre ce principe, il faut voir notre application comme un site web (cela tombe bien, c'est souvent ce qu'essaie de faire Ionic) avec diff rente URL. Nous d finissons dans le module de base de l'application toutes les routes existantes et les lions   un composant pour qu'il sache quoi afficher   tel ou tel appel de route. L'avantage de cette mani re de faire est l'architecture de l'application qui est visible plus facilement et rapidement! Nous pouvons  galement g rer des redirections dans certains cas que nous pouvons programmer (pratique pour une gestion des droits efficaces).

Afin de naviguer entre les pages (les  crans!) nous pouvons dans l'HTML ajouter la directive `routerLink` qui prendra en param tre la route voulu dans les boutons ou dans les balises `<a>` servant de base au web pour naviguer.

La seconde solution repose sur un principe plus orient  application mobile o  l'on y d finie qu'elle est la page racine de l'application et   tel ou tel ouverture de nouvelle page, si celle-ci remplace l'ancienne dans la racine ou si elle s'ajoute dans la pile de page ouverte.

Nous passons via le service `Nav` du paquet Ionic et sommes oblig s pour le d but de d finir un des composant page comme racine de l'application. Pour ajouter une page dans la pile, nous utiliserons la m thode `push` avec en param tre le composant voulu. Pour revenir en arri re,  a sera la m thode `pull`, cela fonctionne comme une liste dans la plupart des langages! La diff rence entre changer la page racine et empiler les pages, c'est que si nous nous trouvons sur une page racine et nous cliquons sur le bouton retour des smartphones Android ou Windows Phone, nous quittons l'application tandis qu'avec une pile, ce m me bouton physique servira

simplement de pull. Le d savantage de cette solution est qu'au niveau du code l'architecture des pages est moins lisible.

L'interaction avec une API

Afin de r cup rer des informations du backend, ou d'y en envoyer, nous utilisons le service HttpClient fourni de base par Angular. Nous y b n ficioons des m thodes get, post, put et delete permettant d'ex cuter ces requ tes vers un web service et nous renvoie un observable avec lequel nous traitons les informations de mani re asynchrone.

Ces m thodes prennent en param tres la route du service web ainsi que de nombreux param tres, comme des headers (dont j'ai besoin pour y faire transiter le token d'authentification) ou un body utile pour envoyer les donn es relatif aux requ tes POST et PUT.

Profitons  galement d'Angular pour utiliser les HttpInterceptor. Ceci nous permet d'intercepter toutes les requ tes vers un web service que nous faisons avant et apr s l'envoi de celui-ci. Dans notre cas, il sera utile en ajoutant   chacune de nos requ tes HTTP un header contenant le token d'authentification de la personne (except  bien  videmment pour la connexion).

Le multilinguisme

Simplicit , c'est ainsi que je d finirais l'internationalisation dans Ionic et dans Angular de mani re plus globale.

Il existe une biblioth que populaire qu'est ngx-translate qui nous facilite grandement la t che.

Les dictionnaires de traductions sont des fichiers json tous plac s dans le m me dossier. Nous nommons chaque fichier de dictionnaire par l'abr viation du code langue IETF (https://fr.wikipedia.org/wiki/Mod%C3%A8le:Code_langue) de la langue utilis . Dans mon cas j'ai donc un fichier fr.json et en.json. Ces fichiers json fonctionnent en cl  valeur, nous d finissons une cl  (qui nous sert d'identifiant unique) qui sera la m me dans toutes les langues et une valeur qui sera la traduction propre   chaque langue.

```
"General": {  
  "Welcome": "Bienvenue",  
  "Add": "Ajouter",  
  "Update": "Mettre   jour",  
  "Cancel": "Annuler",  
  "Confirm": "Confirmer",  
  "AreYouSure": " tes vous sur?",  
  "Error": "Erreur",  
  "InWorking": "En cours de construction",  
  "Date": "Date",  
  "Both" : "Les deux"  
},
```

```
"General": {  
  "Welcome": "Welcome",  
  "Add": "Add",  
  "Update": "Update",  
  "Cancel": "Cancel",  
  "Confirm": "Confirm",  
  "AreYouSure": "Are you sure?",  
  "Error": "Error",  
  "InWorking": "In working",  
  "Date": "Date",  
  "Both" : "Both"  
},
```

Une fois la biblioth que install e, nous devons pr ciser dans le app.module l'emplacement des traductions. Il est conseill  de les mettre avec les assets et faire

un dossier nomm  i18n (Internationalization). C'est une convention et lorsque nous mettons notre application en production, tous les fichiers fixes (images et dictionnaires) seront ainsi au m me endroit.

L'initialisation se fait dans le composant (contr leur) global de l'application, `app.component.ts`, avec le service Translate fourni par la biblioth que, nous lui pr cisons les langues pr sentes dans un tableau de string ("en" et "fr" pour nous), indiquons la langue par d faut (si une cl  manque dans une langue, l'application ira chercher la traduction de la langue mise par d faut).

Pour afficher une traduction, dans l'html il suffit d'y indiquer la cl  voulu avec le pipe "translate"

```
{{ "General.Add" | translate }}
```

Si nous voulons une traduction dans un contr leur javascript, c'est un peu plus compliqu , cela se passe en asynchrone o  nous devons acc der au service Translate,   sa fonction get qui prend en param tre la cl  de traduction voulu (en string). C'est pour moi le seul point noir, devoir passer en asynchrone juste pour r cup rer un string.

Le changement de langue est tout aussi simple, nous appelons une fois de plus le service Translate et lui indiquons que nous utilisons   pr sent une autre langue que celle par d faut en passant le code IETF correspondant. Ainsi, les valeurs changent automatiquement dans l'HTML. Pour les traductions dans les contr leurs, nous devons recharger ces parties manuellement quand un changement est rep r .

Afficher des notifications

Ionic fournit des acc s natif aux propri t s smartphone tel la cam ra, le bluetooth ou les notifications via Cordova.

Dans le cas des notifications, la biblioth que ionic-native fournit directement un module LocalNotifications qui nous permettra de r aliser notre but.

La premi re chose   faire pour afficher   l'utilisateur une notification est de voir si nous en avons la permission (en effet, sur nos smartphone de base les applications n'ont aucun droit natif, c'est   l'utilisateur d'autoriser tel ou tel acc s). Si nous ne l'avons pas, nous pouvons afficher un pop-up le demandant, sinon, nous pouvons commencer   construire l'objet localNotification qui va prendre obligatoirement trois propri t . Premièrement un id servant   le retrouver si nous voulons l'effacer sous conditions. Deuxi mement le texte   afficher et enfin un trigger pour l'enclencher. Mais nous pouvons  galement lui mettre un son sp cifique ou une ic ne particuli re (par d faut cela sera celle de notre application).

Essayons d' tre pr sentable

Si vous  tes un d veloppeur web (ce qui est mon cas), vous serez comme un poisson dans l'eau dans Ionic.

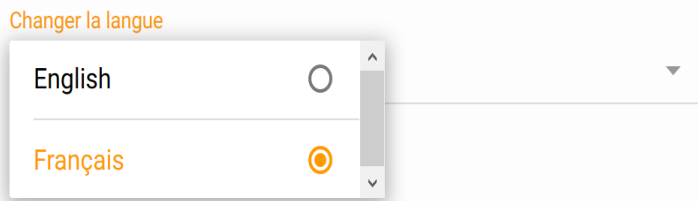
Tout le design se passe via des propri t  CSS classiques, aucun pi ge.

Ionic propose  galement de base une palette de couleurs pr d fini dans des variables pour les  l ments primaires, secondaires, de danger, r ussi, blanc et noir qui automatiquement auront un d grad  et que nous pouvons changer facilement via cet outil <https://ionicframework.com/docs/theming/color-generator> de g n rateur

de couleur nous montrons en direct   quoi cela ressemblerait sur notre application. Autre avantage, les composants d'Ionic se base essentiellement en terme de design sur Material Design, qui est un ensemble de r gles esth tique propos  par Google depuis 2014 et qui sans  tre officiel dans le milieu du mobile,   prit une place consid rable dans le coeur des d veloppeurs mais aussi des utilisateurs.

Ainsi, il est tr s ais  de rapidement proposer un rendu tape   l'oeil sans se prendre la t te et surtout sans  tre un professionnelle du design (ce qui est rarement notre cas en d veloppement)

Voici par exemple une liste d roulante pour s lectionner un item



Ou deux champs de textes, avec le label qui se met   la place du texte tant que celui-ci est vide pour ensuite se d porter au-dessus.

Titre

D tail

Deux Pizzas et une boissons

Ce sont tous des petits d tails rendant la partie design du d veloppement Ionic un plaisir.

Notons tout de m me un d savantage   ce genre de pratique, cela peut amener   ce que toutes les applications se ressemblent trop.

D ploiement

Nous avons deux mani re de faire.

Si nous poss dons Android Studio (ce qui est souvent le cas pour avoir les outils de compilation et d' mulation) nous pouvons ouvrir le dossier Android g n rer par Ionic et de l  compiler notre produit en .apk directement.

L'autre option est de passer via les lignes de commande.

"ionic cordova build android --prod --release" nous permet de g n rer un apk qui se situera dans le dossier "platforms/android/build/outputs/apk". Malheureusement cela n'est pas tout, il faut  galement signer l'ex cutable (sinon la plupart des appareils refuseront de l'installer) en lan ant la commande java "keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 10000". Nous avons d j  trouv  plus intuitif comme proc d .

Autres points importants?

Un avantage d'Ionic dans le développement est qu'on peut également tester notre produit sur un navigateur web. Bien évidemment on ne peut pas tester les accès natifs aux smartphones mais cela s'avère très utile pour voir le rendu graphique implémenté et y apporter des modifications plus rapidement sans passer par un émulateur (et la compilation qui va avec) plus lourd. C'est également un avantage si on souhaite déployer une application smartphone et sa version web!

Ionic propose également des outils payants. Nous avons AppFlow permettant de faire du DevOps en ayant un dashboard pour automatiser des tests avant chaque release, en fournissant un Git et des outils de travaux collaboratif . Il vend également un IDE fait maison (Ionic Studio) où nous pouvons créer une interface sans toucher en code et proposant des facilités propres à Ionic.

Le 23 janvier, Ionic a fait son annonce de la sortie de sa version 4. Comme souvent lors de ce genre d'événement, les performances des outils y sont améliorées et de nouveaux composants sont sortis afin de nous faciliter la tâche. Mais le clou du spectacle fut l'annonce de la prise en charge (en beta pour le moment) des bibliothèques ultra populaires que sont React JS et Vue JS pour ceux qui ne veulent pas d'Angular.

Xamarin

Au d but, Xamarin  tait une entreprise fond e en 2011 qui a cr  e Mono, projet rendant les applications C# multiplateforme (tr s appr ci  par les d veloppeurs Linux et Apple). Et au fur et   mesure de l'avancement du projet, ils ont lanc  Mono Android et MonoTouch (pour iOS) et unifient ces plateformes de d veloppement en 2013 afin de pouvoir avoir des  crans utilisables sur Android et iOS avec le moins de code possible. C'est en 2016 que Microsoft, afin de plus  tre   la tra ne dans le monde du mobile, racheta cette entreprise afin de l'int grer dans Visual Code (pour ne plus avoir besoin de l'IDE maison de Xamarin) et surtout rendre le projet Open-Source!

Xamarin utilise donc C# et le framework .NET de Microsoft afin de faire tourner des applications sur les appareils mobiles. Les habitu es des technologies Microsoft peuvent   pr sent faire des applications smartphones sans apprendre tout un nouveau langage depuis z ro. Pour la partie affichage pure, Microsoft a d velopp  XAML qui tout comme HTML est une surcouche d'XML afin de faire des interfaces graphiques. XAML est aussi utilis  pour faire du d veloppement d'application UWP (Universal Windows Platform) ce qui permet aux personnes ne jurant que par Microsoft de garder leurs marques.

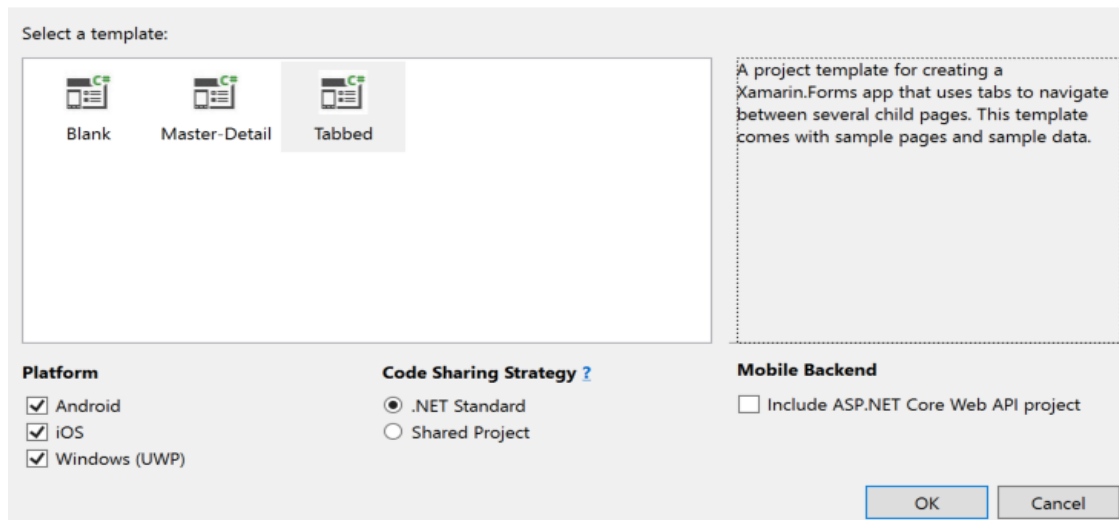
Microsoft recommande d'utiliser l'architecture MVVM, Model-view-viewmodel, bien que nous pouvons faire   notre guise.

Le mod le repr sente un objet avec de r el donn es dedans. La vue elle permet d'avoir un affichage. Quant au viewmodel il sert   automatiser les donn es entre un model et une vue et changer en direct si besoin (nous appelons cela le binding)

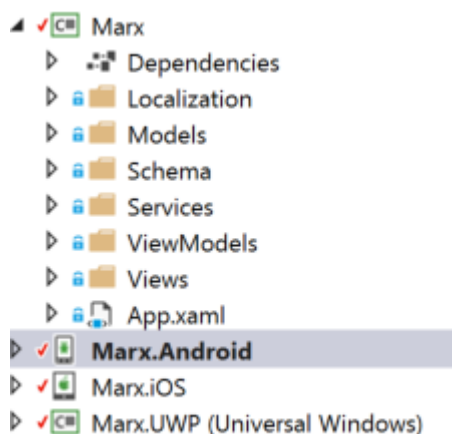
L'initialisation

Microsoft oblige, c'est via Visual Studio que l'on va d marrer le projet. Il est certes possible de le faire ailleurs et jouer avec la CLI ou g rer les fichiers manuellement, mais cela serait une perte de temps. Au lancement du projet, apr s s lection de l'option Cross Platform project, nous pouvons choisir pour quelle plateforme sera notre application (Android, iOS et Windows, il n'est pas oblig  de s lectionner les trois) mais surtout d cider si nous commen ons avec une application vide, une vue avec pagination d'un menu d roulant ou avoir un menu fixe en haut ou bas de page permettant de passer de l'une   l'autre en un clic.

Visual Studio nous propose aussi de lier si nous le souhaitons directement un web service (en ASP.NET, le framework web de Microsoft).



Une fois la solution initialis e, Visual Studio nous fournit un projet de base et un projet par plateforme cibl e. Tout se joue dans le projet racine, les autres  tant l  si nous avons besoin de code sp cifique   une seule des plateformes.



Les dossiers pour respecter le MVVM sont pr sents avec d j  des exemples si nous n'avons pas s lectionner un projet vide.

L'ex cution se fait en choisissant entre un  mulateur Android, Windows Phone, iPhone (non disponible pour moi) ou en mode desktop. Le mode desktop sera souvent utilis  car plus rapide que lanc  sur un  mulateur.

En ce qui concerne la documentation, nous avons   faire   Microsoft et sa documentation fouillis qu'on retrouver sur plusieurs pages diff rentes et souvent incompl te et avec parfois des exemples plus   jours (j'ai eu quelques fois le soucis d'avoir une propri t  dans un exemple n'existant plus!).

Xamarin   un site Xamarin University avec des vid os explicatives et parfois le code source qui va avec. Une section payante est  galement pr sente pour y avoir des certifications ou la possibilit  de parler   des experts pour nous aider.

L'affichage et la gestion d'un formulaire

Le XAML se limite   ce que veut bien nous fournir Microsoft et nous voil  d j  emb t  par les restrictions. Nous n'avons par exemple pas de Check ou Radio Button. Soit nous chipotons en cr ant du code nous m me pour que cela y ressemble soit on esp re trouver notre bonheur dans les biblioth ques fournies par la communaut . Compar    ce que nous fournit de base Ionic, la limitation se fait vite ressortir. Limitation assez incompr hensible   mes yeux non seulement pour une si grosse entreprise qu'est Microsoft mais aussi compar    un projet UWP (pour faire des applications Windows 8 et 10), qui se base  galement sur XAML et qui lui est tr s complet)

Pour la s paration du code, un fichier XAML va se diviser en deux, la partie pure XML o  nous y indiquerons les balises repr sentant nos  l ments et la partie CS (C#, le langage phare de Microsoft) qui lui contiendra le code d'interaction (avec les boutons ou tout simplement l'initialisation de la page) et la liaison avec le vue-model. Concernant le vue-model (qui est un autre fichier CS), on y trouvera les variables utilis es dans l'affichages. Chaque variable public devra avoir son  quivalent en priv  (par convention on y met un '_' devant le m me nom) et les variables public auront aussi un getteur pointant vers son  quivalent priv  afin de s'afficher et un setteur pour modifier son contenu priv . Je trouve ce genre de proc d  fort lourd en d veloppement de devoir ainsi faire des r p titions.

La page en elle-m me se forme dans la balise `ContentPage` qui contient d'abord des propri t s utiles pour les importations (utile pour le multilinguisme) et ensuite la balise `ContentPage.Content` o  nous y disposerons les  l ments. Nous aurons comme choix pour la dispositions de faire des grilles (`Grid`) afin d' tre pr cis dans la r partition visuel soit une pile d' l ment (`StackLayout`) s'affichant les uns en dessous des autres.  voquons aussi l'`AbsoluteLayout` (o  nous fixons les  l ments) et la `ScrollView` qui elle n'est qu'une liste.

Concernant la grille, son positionnement est diff rent d'Ionic o  nous mettons des rangs que l'on d coupe ensuite pour former un rendu responsif. Ici nous d finissons au tout d but de la grille le nombre de colonnes et de rang es et indiquons la taille   chaque fois. La valeur '*' elle permet de prendre toute la place possible et si plusieurs  l ments ont cette valeur, ils se partageront  quitabement cette place.

Apr s avoir d fini notre grille, pour mettre les  l ments   l'int rieur, nous devons leurs indiquer   quelle place de la colonne et de la rang es ils sont avec les propri t s `Grid.Row` et `Grid.Column` (attention comme un tableau on part de z ro et  a sera d'ailleurs la valeur par d faut).

Je n'ai pas l'habitude de ce genre de positionnement. Cela m'a sembl  au d part assez peu intuitif pour voir quel  l ment est   quelle place. Mais finalement cela s'av re assez efficace et pas prise de t te.

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="*" />
    <RowDefinition Height="*" />
  </Grid.RowDefinitions>
  <StackLayout Grid.Column="0">
    <Label Text="Name"/>
    <Entry Text="{Binding Name}" Placeholder="Name"/>
  </StackLayout>

  <Button Text="Add" Clicked="Add_Button_Clicked" VerticalOptions="Start"
    BackgroundColor="{StaticResource Primary}" Grid.Column="1"/>

  <Button Text="Back" Clicked="Back_Button_Clicked" VerticalOptions="Start"
    BackgroundColor="{StaticResource Primary}" Grid.Row="1" Grid.Column="1"/>
</Grid>
```

Faisons maintenant un formulaire. Comme  voqu  un peu plus haut, on remarque d s le d part que nous ne disposons pas de Radio et Check Button pour faire des s lections simples et multiples. Si nous voulons juste s lectionner un  l ment d'une liste il y a le Picker qui est pr sent pour nous (cela affichera un pop up avec la liste voulu) mais cela reste assez pauvre.

Pour une saisie de texte, nous avons la balise Entry auquel on lira (le fameux binding) une variable.

Malheureusement contrairement   Ionic, il n'y a pas d'objet formulaire rendant le tout plus simple. Je ne peux donc de mani re efficace cacher le bouton "Valider" du formulaire tant que les conditions ne sont pas remplies.

Je me contente alors de laisser le bouton tout le temps cliquable et faire les v rifications quand l'utilisateur pressera le bouton en affichant un message d'erreur si besoin.

```
<StackLayout IsVisible="{Binding IsBusy, Converter={Helpers:InverseBoolConverter}}">

    <Label Text="Adding a Relationship"
        VerticalOptions="CenterAndExpand"
        HorizontalOptions="CenterAndExpand" />
    <Entry Text="{Binding Relationship.Name}"
        Placeholder="Relationship type name" />

    <Picker
        Title="Select a relationship type"
        ItemsSource="{Binding RelationshipTypeList}"
        ItemDisplayBinding="{Binding RelationshipType.Name}"
        SelectedItem="{Binding SelectedRelationshipType}"
        />

    <Button Margin="0,10,0,0"
        Text="Add"
        Clicked="Add_Clicked"
        BackgroundColor="{StaticResource Primary}"
        VerticalOptions="CenterAndExpand"
        TextColor="White" />
</StackLayout>
```

La pagination

En partant d'un projet avec un menu, la pagination est d'une grande simplicit . nous avons la page principal qui reprend le menu et un container pour y mettre les pages qui seront choisi via le menu. A la s lection d'une nouvelle page, celle-ci va se mettre en tant que RootPage (objet de la page principal) et faire  galement

changer le titre du menu. Si nous voulons une page qui se met au-dessus de l' cran principal, cela fonctionne comme une pile via l'objet Navigation o  l'on lui pr cise la page   afficher au-dessus de tout. Cette page peut se retirer en l'appelant manuellement toujours gr ce   l'objet navigation ou automatiquement avec le bouton physique retour des t l phones voir encore avec le bouton formant une fl che vers la gauche en haut de l' cran.

Un projet avec les tabulations pour changer de page principal? Encore plus simple! Dans notre page principal nous mettons le XAML TabbedPage avec ses enfants qui seront des onglets avec un titre, une ic ne si besoin et la page correspondante.

```
<NavigationPage Title="Relationship Type">
  <NavigationPage.Icon>
    <OnPlatform x:TypeArguments="FileImageSource">
      <On Platform="iOS" Value="tab_about.png"/>
    </OnPlatform>
  </NavigationPage.Icon>
  <x:Arguments>
    <views:RelationshipTypesPage />
  </x:Arguments>
</NavigationPage>
```

L'interaction avec une API

Le traitement des requ tes se fera dans le view-model d'une page (limitons le code dans la partie CS des fichiers view). Celui-ci appellera le service voulu (qui est une classe   part cr  e par nos soins).

Pour appeler mon webservice j'utilise la librairie (non inclus de base) RestSharp qui me fut conseill  par des coll gues. Celle-ci r duit drastiquement le nombre de ligne   faire (et r duisant aussi par la m me occasion le risque d'erreur). La librairie me fournit l'objet RestClient qui prendra en param tre l'url de base. Ensuite vient l'objet RestRequest o  je peux y inclure le type de m thode (GET, POST, ...) mais  galement le body et le header.

Il me reste   utiliser la m thode Execute de RestClient prenant en param tre le RestRequest con u juste avant et je re ois ainsi une RestReponse contenant la r ponse du webservice.

Il est  galement possible de r aliser tout cela en asynchrone, ce qui est toujours conseill .

Le multilinguisme

Pour rendre son application multilingue, Microsoft   sa biblioth que interne Globalization qui nous y aide.

Nous devons mettre des fichiers 'resx' dans un dossier Localization   la racine du projet. Ce sont des fichiers qu'on peut  diter en xml ou via Visual Studio qui nous met   disposition un  diteur. Comme souvent, cela fonctionne avec une cl  qui sera la m me dans toutes les langues et une valeur qui elle changera en fonction du fichier 'resx' que l'on  dite.

	Name	Value
►	ChooseLang	Changez votre langue
	HelloWorld	Bonjour le monde
	Home	Accueil
	Relationship	Relation
	RelationshipType	Type de relation
*		

A la racine du dossier Views, nous devons cr  er un fichier TranslateExtension o   nous y initialiserons le tout en pr  cisant la langue par d  faut. Et dans chaque fichier XAML de vue o   nous aurons besoin d'une traduction, nous indiquerons dans le ContentPage l'assembly utilis   et le label voulu sera r  cup  r   en indiquant la cl   du dictionnaire.

Attention cependant,    l'affichage Xamarin prendra soit une langue qu'on lui impose soit la langue du t  l  phone si elle est parmi celle que l'on propose. On peut changer en cours d'ex  cution de l'application la langue mais elle ne se changera pas automatiquement partout, il faut    chaque fois recharger les pages d  j   ouvertes avant (ou simplement relancer l'application).

Afficher des notifications

Fonctionnalit   pourtant importante pour une application sur mobile, il n'est pas ais   d'afficher une simple notification avec Xamarin. Tout d'abord, les notifications doivent   tre g  r  es    part pour iOS et pour Android! Ce qui enl  ve l'int  r  t du cross platform. Heureusement nous pouvons comme souvent compter sur la communaut   des d  veloppeurs open-source. Il existe un plugin nuget, "Xam.Plugins.Notifier", qui permet de faire cela tr  s simplement. Il suffit de faire appel    l'objet CrossLocalNotifications, de lui assigner un titre et un corps de message et l'affaire est jou  e! Nous pouvons   galement lui d  finir un d  lai avant d'appara  tre. C'est assez dommage que Xamarin ne propose pas tout simplement cela de base.

Essayons d'  tre pr  sentable

Dans le fichier App.xaml (point d'entr  e de l'application) nous pouvons y d  finir un dictionnaire de cl   / couleur pour r  utiliser dans les   l  ments la m  me couleur (et la changer qu'une seule fois si besoin) et nous pouvons   galement d  finir des propri  t  s globales afin d'  tre utilisable partout. Si nous voulons par exemple que tous nos boutons aient un margin de 10 pixels, nous mettons le code <Thickness x:Key="ButtonMargin">10</Thickness> avec comme cl   "ButtonMargin" et en appliquant    mes boutons Margin="{StaticResource ButtonMargin}" il prend la valeur voulu.


```
<Application.Resources>
  <ResourceDictionary>
    <!--Global Styles-->
    <Color x:Key="NavigationPrimary">#FF9A00</Color>
    <Color x:Key="Primary">#FF9A00</Color>
    <Color x:Key="Secondary">#0B72FF</Color>
    <Style TargetType="NavigationPage">
      <Setter Property="BarBackgroundColor" Value="{StaticResource NavigationPrimary}" />
      <Setter Property="BarTextColor" Value="White" />
    </Style>
  </ResourceDictionary>
</Application.Resources>
```

Je pr f re tout de m me la m thode du web o  en une seule classe permet de d finir plusieurs valeurs. Je suis de plus habit s aux propri t s HTML/CSS mais cela n'est pas non plus insurmontable d'avoir un rendu correct en Xamarin. Le style donn  par d faut aux divers  l ments est dans le rendu standard d'Android ou iOS.

D ploiement

Voil  un point fort de Xamarin. Avec les outils tr s clair de Visual Studio, il suffit de faire un clic droit la solution du projet Android, s lectionner archiver ce qui va g n rer l'APK. Il est  galement possible de signer l'application et de cr er une cl  de signature ais ment via l'IDE. Et enfin, cerise sur le g teau, si nous avons un compte Google pour cela, nous pouvons directement publier notre application sur le Google Play!

Autres points importants?

Un point qui m'a beaucoup d rang  dans mon d veloppement avec Xamarin, c'est le manque de clart  de la documentation.

Premi re, on ne sait pas toujours o  se trouve tous les  l ments. Il y a plusieurs pages de documentation (que ce soit la Xamarin University ou la Doc Microsoft). On ne sait pas toujours o  donner de la t te.

Ensuite, les exemples non   jour. J'ai eu quelques fois des exemples de la documentation officiel qui si je la suivais strictement me faisait des erreurs car tel ou tel propri t  n'existe plus!

Nous ne sommes jamais inform s non plus des importations (use)   devoir utiliser avec certains outils (pour les notifications par exemples).

C'est un soucis que j'ai toujours eu avec les diverses documentations de Microsoft et cela est p nalisant.

Autre point   signaler, Microsoft base sa communication promotionnelle sur le fait qu'en un seul code nous allons avoir une application sur trois plateformes diff rentes (iOS, Android et Windows pour rappel). Parfait, c'est ce que nous cherchons!

Cependant souvent dans la documentation il y a des exemples de codes sp cifiques pour tel ou tel plateforme et ce afin d'avoir une application plus optimis  pour l'un ou l'autre appareil. Ce qui est bien d'un c t  mais respecte plus du coup le principe d'un seul code pour les contr ler tous.

Et au niveau du style, il faut pr ciser certaines choses sp cifiquement dans les dossiers Android et iOS. Les couleurs des menus par exemple, il faut  galement les

faire changer et dans nos fichiers de configuration de base mais aussi préciser les bonnes valeurs dans le fichier colors d'Android. Ce sont des petits détails, mais je trouve cela à nouveau pénalisant.

J'ai également pu tester Visual Studio 2019. J'ai créé un projet Xamarin via cette mise à jour et à présent la version UWP n'est plus prise en charge directement. Cela veut dire que je ne peux plus tester facilement mon application comme un exécutable Windows mais je dois dorénavant d'office lancer un émulateur Android ou iOS. Processus bien plus lourd rendant les tests plus lents!

Cependant, Xamarin à sa version 4 en bêta depuis quelques temps et semble aller dans le bon sens, déjà avec une pagination mieux gérée via des routes (comme avec Ionic) et plus de moyen pour l'affichage. Tout n'est peut-être pas perdu.

Petit élément perturbant à préciser tout de même, Une page en Xamarin s'appelle une "Form". Oui, comme un formulaire en français. Ma recherche de tutoriel pour faire un formulaire en Xamarin fut donc compliqué à cause de cela.

React Native

Tout comme Ionic, React Native se base sur un framework web d j  existant pour former des applications mobiles. Cependant, React Native va puiser dans les ressources de React JS. Ces deux outils sont tous les deux maintenus par Facebook. C'est depuis 2013 que React JS existe, le principe de cette biblioth que est de cr er des composants le plus r utilisables possibles et le but est de pouvoir utiliser React ind pendamment du reste d'une application ce qui veut dire qu'il n'y a pas de r gles ou une structure pr d finie   avoir pour l'utiliser (au contraire d'Angular) afin d' tre le plus flexible possible. C'est  galement pour cela que React JS fonctionne en JavaScript sans la surcouche TypeScript. L'avantage de ceci est que React est exploitable sur un projet d j  existant sans devoir tout refaire, on peut se permettre de l'int grer  tape par  tape. Cela a permis son adoption tr s rapide dans le c ur des professionnelles qui ainsi ne devaient pas repartir de z ro. Comme dit pr c demment, React fonctionne avec le principe des composants, pour en cr er, il nous faut une classe JavaScript  tendant la classe `Component` de base de React. Cette classe aura un `state` qui est l'initialisation des variables (sous forme de json) et un `render`, qui retourne un HTML.

```
super(props);  
this.state = {  
  relationshipTypeName: "",  
  isLoading: false,  
  relationshipTypeList: [],  
  modalVisible: false  
};
```

Contrairement   Ionic et Angular, React ne fait pas de two-way data binding mais uniquement du one-way, cela veut dire que la vue en elle-m me r agit aux changements du "contr leur" mais pas le contraire. Cela est bien  videmment plus performant   l'ex cution de base mais au niveau du d veloppement cela est (selon moi) p nalisant car lors d'un formulaire par exemple, si je veux mettre   jour la variable d'un champ texte, je devrais y mettre un listener qui change manuellement cette variable   chaque rep rage changement.

React met  galement toutes les  l ments d'un composant dans un seul fichier, nous n'avons donc pas un fichier CSS, HTML et JS pour un composant mais un unique fichier JavaScript mettant la logique, le rendu et la style au m me endroit. Cela est voulu car un composant se doit d' tre le plus petit possible.

```
export default class TitleText extends Component {  
  render() {  
    return (  
      <View style={[styles.container, styles.horizontal]}>  
        <Text style={styles.text}>{this.props.text}</Text>  
      </View>  
    );  
  }  
}  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    justifyContent: "center"  
  },  
  horizontal: {  
    flexDirection: "row",  
    justifyContent: "space-around",  
    padding: 10  
  },  
  text: {  
    color: Colors.secondaryColor,  
    fontSize: 20  
  }  
});
```

De nouveau je ne suis pas un fan de ce proc d  car on peut tr s rapidement avoir des fichiers tr s long o  on s'y perd rapidement.

React Native, qui a  t  lanc  deux ans apr s en 2015, va utiliser totalement le m me principe que son grand fr re React JS ce qui va faciliter grandement le plongeon dans le grand bain du d veloppement mobile pour les fans de React JS. La diff rence sera que React Native poss de d j  des composants utilisables par nous et au niveau du style, cela ne reprend pas totalement les m mes propri t s CSS que le web mais tout en restant tr s proches.

L'initialisation

React  tant du JavaScript, c' st  galement via NodeJS que nous devons ex cuter les lignes de commandes afin de mettre en place un projet React Native. La documentation officiel nous propose deux mani res de d buter un projet. Soit nous pouvons passer via l'outil NPM react-native-cli qui va nous mettre un projet sans aucune structure et juste un fichier JavaScript faisant office de premi re page d'application, soit via un autre outil NPM ind pendant, Expo, qui lui nous proposera une base de navigation et trois  crans pr d finis pour montrer des exemples (mais toujours pas de vrai structure). J'ai utilis  Expo pour la suite de cette d couverte car Expo nous fournit des outils en plus pour avoir une application le plus proche possible du natif.

Au premier abord on peut se dire que nous laisser le choix entre deux m thodes d'initialisation est bien (on aime les choix dans la vie g n ralement) mais le soucis qui appara tra rapidement lorsque nous cherchons de l'aide sur internet est que la mani re de faire d'Expo et React-Native-CLI sont diff rentes et certaines applications ou biblioth ques externes ne seront pas utilisables avec l'une des deux m thodes! De plus le manque de structure (une partie pour les pages en tant que soit, une partie pour les composant r utilisable, pour les services, ...) est quelques choses qui m'emb te   chaque fois.

L'affichage et la gestion d'un formulaire

En ce qui concerne la mise en place des divers  l ments sur une page, React fonctionne avec ce qu'on appelle les Flexbox (qui existe  galement dans le web via le CSS). Flexbox est pens  pour ne pas devoir se soucier de la grandeur des  crans, ce qui est pratique dans notre monde rempli d' cran de taille diff rente m me pour les appareils mobiles.

Nous devons d'abord indiquer dans une balise view (qu'on peut comparer  a au Div d'HTML) que le style sera un flex et via la propri t  flexDirection lui indiquer si cela sera un colonne (valeur par d faut) ou une rang e horizontale. Les  l ments   l'int rieur de cette View seront ainsi misent l'un en dessous de l'autre ou l'un   c t  de l'autre. Nous avons ensuite la propri t  justifyContent pour d finir comment se placeront les  l ments   l'int rieur. De base ils seront coll  l'un sur l'autre en partant de la bordure gauche (flex-start) mais nous pouvons les centrer (center), les faire d buter de la droite (flex-end) mais aussi mettre un espace entre chaque  l ments (space-around et space-between).

Voil  ce que nous propose la documentation de React Native et cela s'av re tr s efficace.

Nous pouvons bien  videmment nous en passer et utiliser les m mes types de placement que de Xamarin et Ionic (StackLayout ou Grid).

Tout comme Ionic, React va jouer avec des composants dans tous les sens pour former nos pages et ainsi notre application.

Pas de grand d paysement donc, except  que, nous devons importer   chaque fois tous les composants utilis s... M me les plus basiques et natif   React tel le composant Button, Text, View, StyleSheet (pour notre CSS), ce qui fait que d s le d part, nous aurons une page avec beaucoup d'importation. Et bien  videmment attention   ne pas les oublier du coup sinon l'application ne fonctionnera pas, ce qui pour moi habit  d'Angular m'a jou  beaucoup de mauvais tour.

React utilise donc le One-way-data-binding pour afficher des variables, ce qui veut dire que la vue r agira au changement du contr leur, mais pas l'inverse.

Pour afficher une variable, nous l'initialisons dans le contr leur en le passant dans l'objet state. Et dans la vue, il suffira d'aller le chercher toujours dans cet objet state!

En ce qui concerne les m thodes qu'on veut lier   une action, cela reste tout aussi basique. Nous cr ons donc la fonction dans le contr leur puis si un composant poss de une action (Button et son action onPress) nous faisons une fonction lambda qui appellera la m thode du contr leur voulu. Nous pouvons  galement  crire la fonction directement ici mais ce n'est pas propre niveau pratique de d veloppement.

```
<Button
  icon="add-box"
  mode="contained"
  onPress={() => {
    this.setModalVisible(true);
  }}
/>
```

Jusque-l , tout est simple et m me agr able   utiliser. Mais le One-way-data-binding devient limit  en terme de confort quand on veut faire un formulaire car bien souvent on voudrait bien que le contr leur se met   jour tout seul quand l'utilisateur rentre des informations. Cela est utile pour valider les donn es  crites et autoriser tel ou tel action (bien souvent appuyer sur le bouton valider qui enverra le formulaire au backend). Pour r gler cela nous allons mettre dans les composant de formulaire (TextInput pour un texte par exemple) l'action onChangeText qui appelle la m thode setState (m thode toujours pr sente d s que notre composant   un state avec des variables dedans) qui avec en param tre son variable que l'on va changer va mettre   jour le state.

```
<TextInput
  label={i18n.t("General.Name")}
  value={relationshipTypeName}
  onChangeText={relationshipTypeName =>
    this.setState({ relationshipTypeName })
  }
/>
```

Et pour l'exemple du bouton cliquable uniquement quand le formulaire est valide, nous devons dans la propri t  disabled du composant Button lui indiquer les r gles ou passer via une m thode lui disant si c'est valide ou non, m thode qui sera appel    chaque changement dans le formulaire.

Ceci n'est pas sp cialement compliqu , mais c'est une  norme perte de temps de devoir   chaque formulaire jouer ainsi avec le state! Dans mon exemple d'application j'ai des formulaires de un ou deux champs, aucun soucis. Mais j'ai  galement un formulaire pour enregistrer un paiement qui contient six champs requis et je vais donc devoir pour chaque champs mettre en place le changement de la vue vers le contr leur. Selon moi, ce genre de pratique est le plus gros point faible de React et m'emp che de pleinement profiter du framework de Facebook.

La pagination

En ce qui concerne la navigation, React Native va encore puiser dans ce qui a d j   t  fait avec React JS et utiliser une biblioth que commune, React Navigation.

Celui-ci va fonctionner d'une mani re semblable   ce que l'on fait avec Angular et Ionic. Nous d finissons un "StackNavigator" qui contient un json avec comme clefs chaque nom que nous voulons donner   nos routes (servant d'id) et en valeur   quel component chaque nom est li .

Ceci devra  tre fait dans le component initiateur de l'application (souvent nomm  App) et servira ainsi de conteneur pour tous les composants li s aux diverses routes.

L'interaction avec une API

Contrairement   Angular, React ne propose pas d'outils sp ciaux pour faire appel   un webservice. Cependant JavaScript a d j  ce qu'il faut pour cela et React ne cherche pas   r inventer la roue   tout prix.

Il est ainsi assez ais  de chercher   communiquer avec un serveur distant en JavaScript via un objet fetch, nous devons de base uniquement lui donner un string comme param tre qui est l'url que nous souhaitons appeler. Par d faut cela fera une

requ te GET. Comme second param tre nous lui passerons un objet avec comme propri t  method qui sera GET, POST, PUT ou DELETE, un body si il faut (dans le cadre du POST et PUT) ainsi qu'un headers.

Ce Fetch peut s'ex cuter de mani re asynchrone (primordiale de nos jours) et nous pouvons ensuite transformer notre r ponse en json.

```
export const getRoute = async route => {  
  try {  
    const bearer = await AsyncStorage.getItem("Bearer");  
    return fetch(`${baseUrl}${route}`, {  
      method: "GET",  
      headers: {  
        Accept: "application/json",  
        "Content-Type": "application/json",  
        Authorization: `Bearer ${bearer}`  
      }  
    }).then(response => {  
      if (!response.ok) {  
        verifyStatut(response);  
      }  
      return response.json();  
    });  
  } catch (error) {  
    // Error retrieving data  
  }  
};
```

Le multilinguisme

Continuons les comparaisons avec Ionic en parlant du multilinguisme car il est   peine plus difficile   mettre en place qu'avec Angular.

Tout d'abord, nous aurons   nouveau des fichiers Json qui seront nomm s chacun avec l'abr viation de la langue voulu (fr, en, nl, de, ...) et des paires cl s - valeurs pour aller chercher les traductions. Jusqu'ici nous sommes en terrain connu.

Lors de mes recherches pour un outil React me permettant de g rer les langues, c'est le module NPM i18n-js qui est souvent ressorti.

Pour l'initialiser, il suffit de lui dire quelle fichier de langue nous voulons qu'il aille chercher dans le dossier "lang" et ensuite toutes les abr viations possible. Pour rep rer la langue de l'utilisateur Expo a un objet Localization nous permettant de le retrouver avec la propri t  locale. Tout cela est   faire dans le render() de l'application de base

```
render() {  
  i18n.locale = Localization.locale;  
  i18n.translations = { en, fr };
```

Pour ensuite ressortir la traduction voulu, dans un composant de vue nous devons re importer i18n-js (petit point faible par rapport   Angular o  nous ne devons pas nous soucier de cela) et dans le render faire appel   la traduction voulu via la cl s qui

nous retournera la valeur du fichier Json de traduction de la langue enregistr  dans le `i18n.local` vue avant.

```
<Text style={styles.helpLinkText}>  
  {i18n.t("General.HelloWorld")}  
</Text>
```

Si nous voulons changer la langue en cours d'ex cution de l'application, cela devient un peu plus complexe. En effet on ne peut pas changer la langue depuis n'importe quel composant mais uniquement depuis le composant conteneur (pour rappel, on l'appelle par convention `App`). Il faudra donc   chaque fois faire appel   celui-ci via une m thode qu'on peut atteindre.

Afficher des notifications

 a sera via `Expo` que nous pouvons afficher des notifications. Pour ce faire nous aurons besoin de ses outils `Permissions` (afin de savoir si l'utilisateur nous autorise   ce qu'on lui montre des notifications sur son appareil Android) et `Notifications`. Avant donc de mettre en place l'objet `Notifications` nous allons v rifier si nous avons ces droits d'acc s natif. Si ce n'est pas le cas, nous pouvons les demander.

Essayons d' tre pr sentable

Pour modifier le style de notre application, `React` va se rapprocher de ce qu'on fait sur le web avec des petits changements. Nous aurons dans le composant une variable qui aura l'objet `StyleSheet.create` avec en param tre un json de cl  pour le nom de la classe de style et comme valeur un autre json avec les param tres de style. Il est bien  videmment possible d'avoir un style global mais avec `React` on adore jouer le plus possible avec les composants.

Pour qu'un  l ment du DOM adopte le style voulu, on lui met la propri t  "style" qui sera le nom de la variable de style et la cl  voulu.

Concernant les param tres de style modifiable, alors qu'en CSS on utilise le kebab-style (`font-size`, `background-color`), ici on utilisera le camelCase (`fontSize`, `backgroundColor`).

Si nous sommes habitu s au style du web nous ne serons pas perdus.

Il n'y a malheureusement pas dans la documentation officielle un r pertoire de toutes les valeurs de style changeable.

D ploiement

"expo publish", peut-on faire plus simple que cette commande    crire pour avoir notre ex cutable Android?

Il faudra ensuite faire signer l'APK qui a  t  g n r  mais cela est du ressort de Google et non de `React Native`, je vous invite   relire la partie d ploiement d'`ionic` car cela sera exactement la m me m thode.

Autres points importants?

Je n'ai plus de points significatifs   relever concernant `React`. Durant la r daction de ce travail il n'y a pas eu de grande annonce ou nouveaut  pr vue.

React Native et JS va continuer à être apprécié par de nombreuses personnes, probablement être de plus en plus utilisé car cela reste un outil pratique pour faire du multiplateforme.

Saluons tout de même le site de React Native avec un très bon guide pour prendre en mains ce framework et surtout des exemples que l'on peut changer en direct sur le site afin de voir rapidement ce que peut donner tel ou tel propriété!

Comparaisons r capitulatif

Je vais ici pr senter sous forme de tableau les points forts et les points faibles de chaque frameworks, le tout ayant bien  videmment une part de subjectivit .

En voici le code couleur:

Tr s bon, agr able   exploiter

Rien de sp cial   redire

J'y ressens des lacunes qui m'ont ennuy 

Catastrophique selon moi, cela m'a d rang 

	Ionic	Xamarin	React Native
Mettre les outils en place			
Initialisation d'un projet			
Qualit� de la documentation			
G�rer la mise en forme d'une page			
Afficher une liste d'�l�ment			
R�diger un formulaire			
La pagination			
Communiquer avec un Webservice			
Multilinguisme			
Les notifications			
Jouer avec le style			
D�ployer l'application			

Conclusion

Apr s ce long travail de veille technologique, je ressors assez d cu du d veloppement hybride, il n'y en a pas un qui me fait dire "Il faut absolument l'utiliser! Il va  craser toute concurrence!", ils ont chacun d'assez gros d faut que ce soit en terme de performance ou de d veloppement. Mais ils ont aussi chacun des choses plaisantes   proposer qui me fait garder espoir dans l'avenir de ce type de d veloppement!

Au final j'en ressors avec deux conclusions

D velopper sp cifiquement pour mobile est-il toujours utile?

Non! Quand nous faisons un site web de nos jours, une attention particuli re sur le responsive design (le fait d' tre visible correctement sur tout type d' cran) est de plus en plus souvent port . Et tr s souvent cela est suffisant pour les besoins des utilisateurs finaux. Qu'apporte les OS mobiles en plus? La possibilit  d' mettre des notifications? C'est le nouveau spam depuis que les mails ne sont plus   la mode (et les navigateurs web permettent cela aussi maintenant). La localisation? Elle est  galement disponible sur le web.

Le meilleur exemple selon moi sont les applications des m dias journalistiques. Qu'avons-nous besoin pour une telle application? Savoir lire des textes, observer des images et parfois des vid os pour plus d'explication. Un navigateur internet permet il cela? H  bien oui.

Je ne dis bien  videmment pas que d velopper distinctement pour mobile est une mauvaise chose, certaines applications sont moins user friendly dans leurs variantes navigateur web (je pense   Instagram pour poster directement une photo ou Skype pour faire des conf rences vid os) mais il est important de se poser la question avant chaque d but de projet si oui ou non le d veloppement mobile nous apporte un plus que si nous nous contentons du web.

Car au final, le vrai multiplateforme n'est-il pas le web tout simplement?

D velopper en pure natif pour Android et iOS n'est pas toujours une perte de temps

Avant toute chose, rappelons comment se d roule le processus de cr ation d'une application.

Une fois l'id e bien d fini, une analyse se fait pour ressortir tous les besoins des fonctionnalit s qui seront d velopp es sans oublier la base de donn es.

Le backend pour interagir entre nos t l phones doit se faire  galement.

Ainsi que des maquettes   dessiner pour d finir   quoi ressemblera la partie visible de l'application.

Tout ce que je viens de citer avant, qu'on développe une application hybride ou natif, ne sera réalisé qu'une seule et unique fois.

On se rend compte dès lors que la programmation de l'interface ne prend pas autant de temps qu'on pourrait le croire! Les applications mobiles étant souvent utile que pour une seule tâche à réaliser.

Rappelons également qu'une application hybride sera moins performante que le natif. Nos smartphones actuel ont beau souvent être plus puissante qu'un ordinateur, il reste beaucoup de téléphone portable milieu ou bas de gamme et l'optimisation doit rester importante.

Les développeurs derrières les frameworks présentés peuvent être les meilleurs du monde, traduire du code JavaScript ou C# vers du Swift ou du Java sera toujours moins optimisé que développer directement en Swift ou Java justement.

Concernant les jeux vidéos sur mobile (qui sont de grosse application donc), il existe des moteurs de jeux (framework spécialisé en jeux vidéos) conçu pour cela et faisant en sorte que cela soit directement multiplateforme et optimisé (nous pouvons citer Unity de Microsoft par exemple)

Scénario de recommandation

Android et iOS

Je suis dans une grosse boite avec beaucoup de développeurs à disposition, je peux donc me permettre de faire deux développements séparés afin d'avoir plus de performance.

C'est selon moi le scénario idéal et ayant pu côtoyer des développeurs et managers mobiles où j'ai travaillé, c'est le scénario que les entreprises tentent d'avoir le plus souvent même si ce n'est pas toujours facile et que si c'est lié à un client il vaut mieux lui expliquer que cela prendra plus de temps et surtout risque d'être plus cher.

Xamarin

J'ai une équipe de développement purement Microsoft dont je ne veux pas perturber les habitudes.

J'aurais du mal à recommander à quelqu'un de neutre niveau technologie d'utiliser Xamarin. La documentation et les guides qui y sont lié sont désastreux et pas à jour. Il y a des manques au niveau de la gestion d'une interface. Et au final Xamarin pousse parfois à faire du code indépendant pour Android et iOS ce qui va à l'encontre de ce que l'on recherche. Mais tout n'est pas à jeter dedans! Et les fans de Microsoft pourront y trouver leurs bonheur!

Ionic ou React Native

Soit j'ai une petite équipe ou moins de temps que prévu et je dois absolument fournir en même temps une application et sur iOS et sur Android.

Soit j'ai une équipe web performante qui veut un peu de changement alors ce choix sera le bon également.

On prendra bien évidemment Ionic si l'équipe est habitué à Angular et React Native si elle utilise usuellement React JS.

Et du coup? Mon choix: Ionic

Tout n'est bien  videmment pas   jeter dans le d veloppement hybride.

Il y a pas mal de cas de figure o  utiliser un des frameworks pr sent s sera une b n diction comme le montre mes sc narios.

La bataille des langages en g n ral est aussi une question de go t et d'habitude. J'ai commenc  ma carri re professionnelle dans le monde JavaScript, j'ai appr ci  ce langage via TypeScript et cela m'avantage de pouvoir l'utiliser pour un projet d'application mobile.

Ainsi mon choix se porterait donc sur l'utilisation d'Ionic et lors de mes projets personnelles,  a sera toujours vers lui que je me tournerais.

React Native (et React JS tout simplement) me plait beaucoup moins. Je dirais m me que je ne comprends pas du tout l'engouement autour de cela. Mais il faut reconnaître qu'il est bien plus performant que ses concurrents.

En ce qui concerne Xamarin... Il a de trop grand manque pour que je puisse le recommander   quelqu'un, de plus j'ai l'impression qu'on doit se compliquer la t che par rapport   ce que j'ai l'habitude.

Je pourrais faire la conclusion bateau de ce genre de comparaison: "Il n'y a pas un framework meilleur qu'un autre, il faut choisir en fonction des besoins, de l' quipe et des go ts". Elle est en partie vrai et raisonnable. Cependant je pr f re insister sur la n cessit  de bien r fl chir au besoin mobile de l'application, ne pas  carter trop vite de le faire en web et si le choix se porte tout de m me sur un framework hybride, je vous recommande Ionic, le plus ancien, qui a le plus fait ses preuves et qui apporte la meilleure p rennit  avec son mode de d veloppement sorti d'Angular qui a selon moi la meilleure lisibilit , ce qui est primordial sur les projets   long terme. De plus, sa documentation est solide, accompagn  de petit tutoriel et son futur me semble radieux en proposant r guli rement des mises   jours.

Source

Code source du projet

Backend: https://github.com/kingdomflo/M-A-R-X_Backend

Frontend: https://github.com/kingdomflo/M-A-R-X_Frontend

Langage - biblioth ques utilis es ou  voqu es

PHP: <http://php.net/>

Framework PHP Lumen: <https://lumen.laravel.com/docs/5.7>

Doc de l'ORM Eloquent: <https://laravel.com/docs/5.7/eloquent>

Auth0: <https://auth0.com/#/>

Auth0 avec Ionic 3: <https://auth0.com/docs/quickstart/native/ionic3>

Ionic: <https://ionicframework.com/> <https://github.com/ionic-team/ionic>

Token JWT: <https://jwt.io/>

Librairie PHP pour JWT: <https://github.com/lcobucci/jwt/blob/3.2/README.md>

Librairie NodeJS pour JWT: <https://github.com/auth0/node-jsonwebtoken>

Composer: <https://getcomposer.org/>

NodeJS: <https://nodejs.org/en/>

Bootstrap: <https://getbootstrap.com/>

Material design: <https://material.io/design/introduction/#principles>

RestSharp: <http://restsharp.org/>

Xamarin plugin notification: <https://github.com/edsnider/LocalNotificationsPlugin>

Expo pour React: <https://expo.io/>

React Native: <https://facebook.github.io/react-native/>

Fetch: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

React Navigation: <https://reactnavigation.org/>

i18n-js: <https://github.com/fnando/i18n-js>

Outils utilis s

Postman: <https://www.getpostman.com/>

Visual Code: <https://code.visualstudio.com/>

Diagramme UML: <http://plantuml.com/>

JetBrains: <https://www.jetbrains.com/>

ADM: <https://developer.android.com/studio/run/managing-avds>

Autre

Popularit  de JavaScript en 2018: <https://2018.stateofjs.com/>

Part de mach  des OS mobiles:

<http://gs.statcounter.com/os-market-share/mobile/worldwide>

Framework existant:

<https://blog.jscrambler.com/10-frameworks-for-mobile-hybrid-apps/>

<https://www.weboptimization.com/blog/hybrid-mobile-app-frameworks/>

JQuery: <https://jquery.com/>

Xamarin University: <https://university.xamarin.com/>

Stackoverflow: <https://stackoverflow.com/>