

Charles EDOU NZE

DÉVELOPPEMENT D'APPLICATIONS MOBILES AVEC IONIC

Créez rapidement des applications multiplateformes, robustes et intuitives

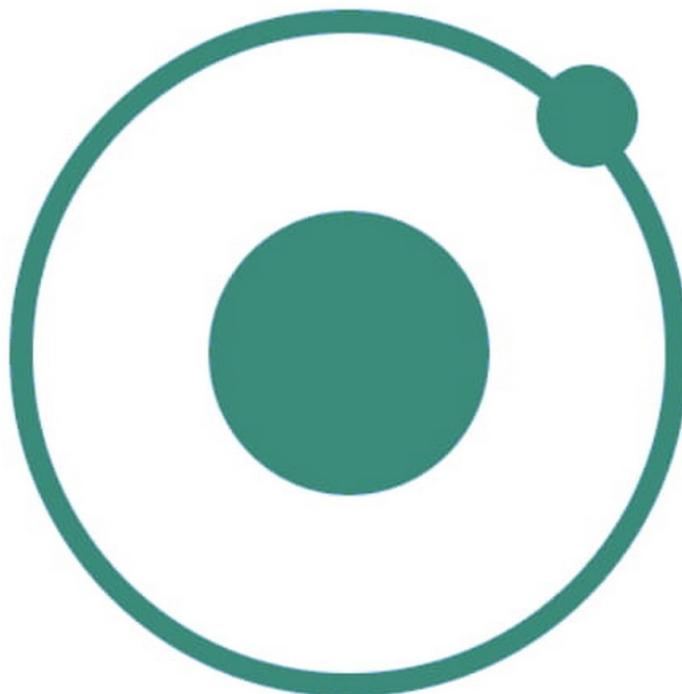


Table des matières

Copyright	1.1
A propos de l'auteur	1.2
Introduction	1.3
Chap 1 - Une brève histoire du mobile	1.4
Chap 2 - Cas pratique : DuckCoin, la cryptomonnaie sur mobile	1.5
Chap 3 - Installation de Ionic et première prise en main	1.6
Chap 4 - Templates et Customisation	1.7
Chap 5 - Utilisation des composants Ionic	1.8
Chap 6 - Introduction au langage TypeScript, le futur de JavaScript	1.9
Chap 7 - Introduction au framework Angular	1.10
Chap 8 - Architecture avancée d'une application Ionic : Composants, Directives, Providers, Services, Pipes, Modules, persistance de données et plugins natifs	1.11
Chap 9 - Tests et débogages avancés	1.12
Chap 10 - Ionic et son écosystème : Cloud, Lab, View et Creator	1.13
Chap 11 - Publication sur les stores	1.14
Chap 12 - Introduction au PWA avec Stencil et Capacitor	1.15
Couverture - Développement d'applications mobiles avec Ionic - Charles EDOU NZE	1.16

Copyright

Le présent ouvrage sous sa version numérique est placé sous licence MIT. Cette licence donne à toute personne recevant cet e-Book le droit illimité de l'utiliser, le copier, le modifier, le fusionner, le publier, le distribuer, le vendre et de changer sa licence. La seule obligation est de mettre le nom du ou des auteurs avec la notice de copyright.

De plus, les exemples cités dans cet ouvrage sont présentés à titre d'illustration et l'auteur ne recommande pas leur utilisation dans le cadre d'applications destinées à un usage public ou professionnel.

Cet ouvrage existe également en [version PDF](#) avec les mêmes droits cités précédemment.



Pour tout renseignement, merci de contacter l'auteur directement depuis son site internet (menu contact) ou via les réseaux sociaux :

- **Site internet** : <https://charlesen.fr>
- **Twitter** : <https://twitter.com/charlesen7>

A propos de l'auteur



Charles EDOU NZE

Ingénieur d'études et développement informatique, je travaille depuis plusieurs années sur les technologies liées au web et aux mobiles (smartphones, systèmes embarqués), aussi bien dans le cadre de projets professionnels que personnels.

Passionné de nouvelles technologies, je sais m'adapter aux changements dans le domaine des NTIC et m'efforce de rester en phase avec ce qui se fait de mieux aujourd'hui. Ce livre est d'ailleurs motivé par ce besoin constant de m'adapter aux évolutions en la matière : rien de mieux pour consolider ses acquis que de les partager avec le plus grand nombre. Il y a en effet plus de bonheur et de bienfaits, à donner qu'à recevoir.

Pour plus d'informations, n'hésitez pas à visiter mon site internet, vous pourrez m'y contacter directement pour des retours, des corrections à apporter ou tout simplement discuter de la pluie et du beau temps.

Charles EDOU NZE,

Site internet : <https://charlesen.fr>

Introduction

Dès les premiers instants du mobile, il n'y avait vraiment qu'une seule façon d'offrir aux utilisateurs la performance et les fonctionnalités qu'ils attendaient : vous deviez utiliser un SDK, c'est à dire un ensemble d'outils logiciels spécifiques à la plateforme que vous souhaitez cibler. Bien sûr, cela impliquait un certain nombre de contraintes :

- Construire une version différente pour chaque plate-forme mobile
- Gérer plusieurs codes sources
- Embaucher et retenir des développeurs natifs hautement spécialisés et coûteux

Depuis, beaucoup d'eau a coulé sous les ponts et les demandes de produits adaptés aux mobiles ont augmenté de manière exponentielle.

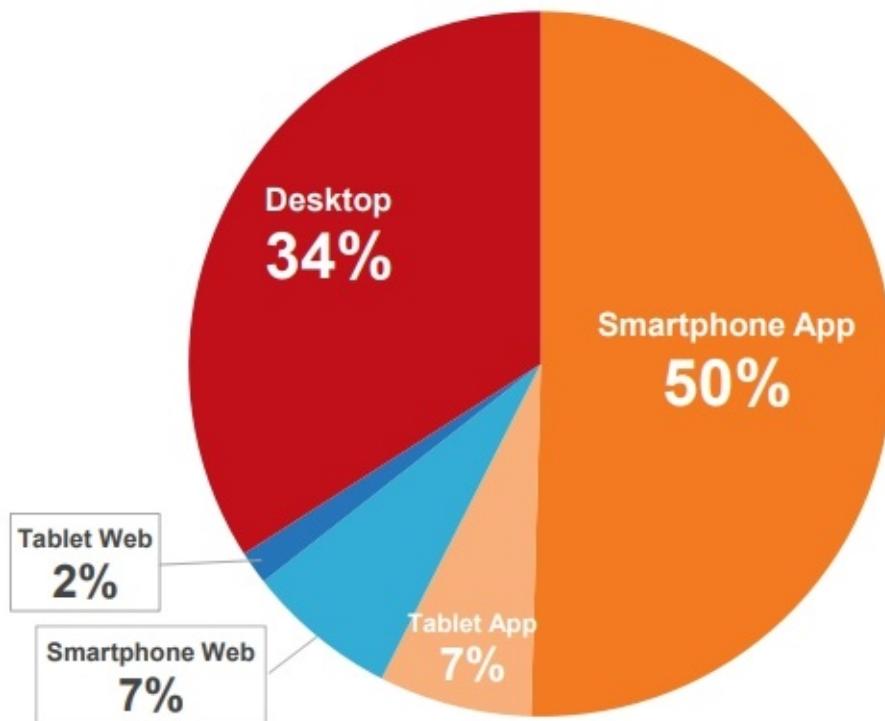
Ainsi, selon un rapport¹ de l'entreprise américaine comScore (analyse publicitaire), le nombre d'utilisateurs mobiles a dépassé les utilisateurs d'ordinateurs de bureau en 2014.

L'entreprise a réalisé aux États-Unis une étude qui permet de mieux connaître les habitudes des internautes sur mobile.

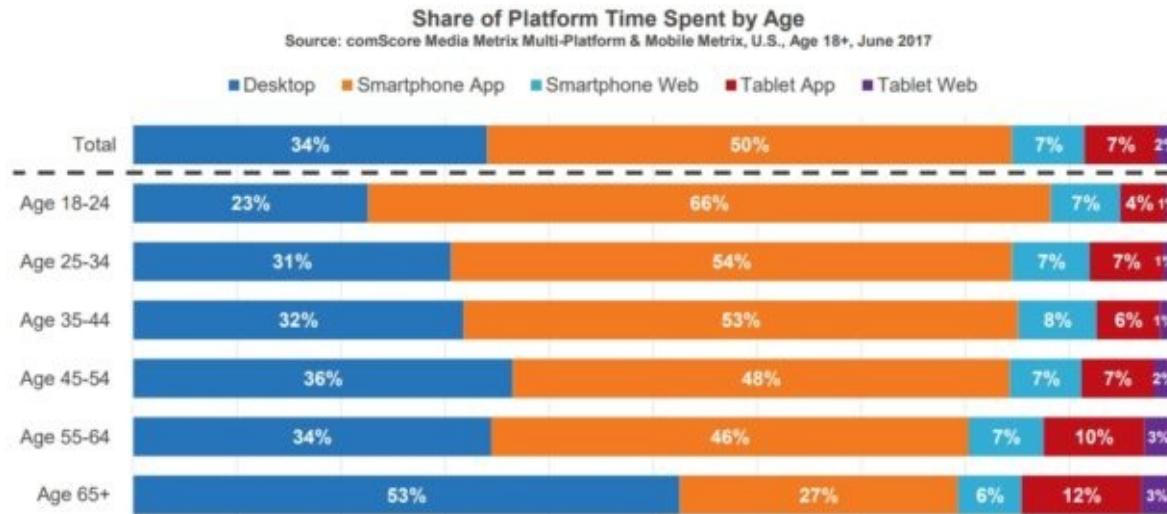
On apprend ainsi que le mobile représente deux tiers du temps passé en ligne, le desktop (ordinateur) ne représentant plus que 34% du temps digital.

Share of Digital Media Time Spent

Source: comScore Media Metrix Multi-Platform & Mobile Metrix, U.S., Total Audience, June 2017



Les 65 ans sont ceux qui consacrent le plus de temps aux ordinateurs. À l'inverse, les 18-24 ans passent moins d'un quart de leur temps sur un ordinateur...pour mieux le consacrer au mobile.



Globalement, et la plupart des études statistiques sont unanimes là-dessus, les 18 à 24 ans sont ceux qui utilisent le plus les applications mobiles. Ils sont suivis de près respectivement par les 25 à 34 ans, les 35 à 44 ans, les 45 à 54 ans, puis les plus de 55 ans. Les femmes passeraient plus de temps que les hommes sur le web mobile, et la moyenne de temps passée sur des applications mobiles a augmenté de 20% en seulement 1 an, entre 2014 et 2015.

Selon une autre entreprise américaine, Gartner, plus de 268 milliards de téléchargements mobiles devaient générer des revenus d'environ 77 milliards de dollars en 2017.

De plus, les applications mobiles ont engrangé des bénéfices atteignant les 35 milliards de dollars en 2014, 45 milliards de dollars En 2015, 58 milliards de dollars en 2016, et comme dit précédemment, un montant prévu de 77 milliards de dollars en 2017. Autant dire qu'il y a énormément d'argent et surtout d'opportunités en jeu.

D'ici 2022 par exemple 70% de toutes les interactions de logiciels d'entreprises devraient s'effectuer sur des appareils mobiles. Donc oui, il y a là un gros marché à exploiter, et qui ne cesse de croître au fil des années.

S'il y a quelques milliers d'années, l'écriture a été l'un des déclencheurs d'une révolution qui nous impacte encore aujourd'hui, il faut croire qu'être douée en développement mobile est aujourd'hui, et le sera encore durant longtemps, une compétence à posséder d'urgence.

Il y a quelques années lorsque j'ai commencé à m'intéresser au développement d'applications mobiles, je n'avais que quelques notions en développement informatique. Mes études m'avaient permis de toucher à des langages de programmation comme Java, le PHP, le Pascal ou encore le C. Je faisais un peu de web, et j'ai compris à un moment donné qu'il était temps de passer à un niveau au-dessus. M'adapter aux tendances du moment. Oui voilà je me parlais à moi-même, du genre : *"Allez charly ne râte pas l'occasion de faire ton entrée dans l'histoire de la tech!"*.

A cette époque, la tendance était très portée sur le natif et les choix proposés n'était pas forcément les meilleurs. Créer une application mobile, même basique, relevait presque toujours du parcours du combattant, quand cela était tout simplement impossible pour le commun des mortels.

Aujourd'hui la donne n'est absolument plus la même et tant mieux d'ailleurs. Tout le monde peu plus ou moins développer, sans trop de mal, une application mobile simple en seulement 24h et la publier sur le Google Play Store. J'ai d'ailleurs un ami, artiste professionnel, qui va bientôt lancer son application mobile à destination d'étudiants en musique. Le plus drôle est qu'il va le faire avec un niveau quasiment nul au départ en développement web et mobile.

C'est pour dire à quel point les choses ont évolué. Ce qui était, il y a encore quelques années, réservé aux professionnels du développement, les fameux **"Ninja"**, est désormais beaucoup plus accessible aux jeunes développeurs ou futurs d'entre eux.

Je ne peux m'empêcher de penser à cette époque où il fallait être un véritable "génie" de l'informatique pour mettre en place, puis publier un site internet sur le web. Aujourd'hui, quelques clics suffisent pour le faire grâce notamment aux CMS (Content Management System pour Système de Gestion de contenus) comme Wordpress ou Joomla!.

Si la porte d'entrée des stores étaient autrefois réservées à quelques privilégiés (les habitués quoi), tout le monde peut aujourd'hui se lancer à l'assaut du gros marché du mobile et pourquoi pas devenir la prochaine licorne du milieu high-tech.

Si vous lisez ce livre en ce moment c'est que vous avez décidé comme d'autres avant vous de vous lancer dans ce monde merveilleux du développement d'applications mobiles. Vous rêvez de grandes choses, vous avez des idées qui sont susceptibles de révolutionner la vie de milliards d'individus, et vous êtes conscient qu'avoir des compétences en développement mobile vous sera utile : bravo, ce livre a été écrit pour vous.

¹. *The 2017 US Mobile App Report*, à [télécharger en version PDF](#) : ↵

Chap 1 - Une brève histoire du mobile

Il était une fois...

Il y a longtemps, fort fort longtemps, en année technologique bien sûr, le natif était roi et les développeurs d'applications mobiles ses sujets. Les temps de développement étaient en moyenne estimés en mois, la rémunération était attractive et les meilleurs développeurs étaient des demi-dieux. Nul ou presque ne pouvait prétendre proposer une première version, la tant convoitée V1, en moins d'une journée sans passer pour un illuminé, pire un hérétique.

Je vous l'ai dit, c'était il y a fort longtemps.

Vous vouliez afficher du contenu issu d'une API (rien à voir avec le dieu égyptien) depuis votre BlackBerry Bold 9000 ? ou une petite image toute mignonne et quelques lignes de texte sur un HTC Dream tout neuf ? Pas de soucis, pour l'un comme pour l'autre, il vous fallait juste maîtriser le Java.

Si vous aviez le malheur de vouloir visualiser vos articles de blog et ceux de vos pages préférés depuis votre iPhone 3G en passant par un flux RSS créés par vos soins, il vous fallait absolument connaître l'Objective-C. Sinon, à moins d'avoir un gros budget et embaucher un développeur mobile, vous seriez contraint de passer votre chemin et plus vite que ça !

Avec le temps, de nombreux projets ont vu le jour pour permettre à tout le monde de se lancer dans l'univers impitoyable du développement d'applications mobiles, avec des technologies classiques du web (html, css, javascript). Beaucoup sont morts-nés, d'autres ont dominé le monde un temps avant de s'éffondrer. D'autres encore, d'abord tapis dans l'ombre, ont su se démarquer, se réinventer, s'adapter jusqu'à atteindre leur plein potentiel.

Une méthode pour les gouverner tous

Il existe différentes façons de développer un produit à destination des mobinautes : le web mobile, le natif et l'hybride.

Applications web mobiles

Une application web mobile est une version allégée d'un site internet, optimisée pour les écrans de petites tailles (smartphone, tablette, TV connectée,...). On parle la plupart du temps de site responsive, car il s'adapte aux caractéristiques de l'écran du navigateur web. Avec le [PWA, concept abordé au chapitre 12](#), les applications web mobiles sont en passe de dominer le trafic issus des appareils mobiles.

Applications natives

Une application mobile est dite native, si le développement effectué pour la créer (et donc le langage de programmation) est spécifique à la plateforme cible.

Plateforme cible	Langage de Programmation
Android (Google)	Java, Kotlin
iOS (Apple)	Objective-C, Swift
Windows Phone (Microsoft)	C-Sharp (C#)
...etc	

Applications hybrides

Vous vous rappelez du seigneur des anneaux ? Oui ? Et bien les technologies dites "**hybrides**" sont au mobile, ce que l'anneau de Sauron est à l'univers imaginée par Tolkien¹.

Une application est dite **hybride** si elle est développée pour les smartphones avec les outils classiques du web.

Les applications mobiles hybrides permettent aux développeurs de réutiliser leurs compétences existantes en développement web. Car, c'est bien connu, les développeurs n'aiment pas être bloqués par des contraintes (maîtriser Java, avoir absolument un Mac ou un PC windows,...) imposées par des plates-formes propriétaires.

Le développement d'applications mobiles hybrides est aujourd'hui la méthode la plus attrayante pour la rentabilité d'une organisation. Pourquoi embaucher un développeur pour chaque plate-forme lorsque vous pouvez embaucher un développeur et les cibler toutes grâce à des technologies aussi connues que HTML, CSS et JavaScript?.

Les smartphones Android, iOS ou autre, embarquent dans leur noyau une technologie appelée **WebView** permettant d'exécuter du code web au sein d'un environnement natif. Ainsi, il est possible de lancer la caméra de votre téléphone (natif), via une simple fonction Javascript (web). Ce qui donnerait ceci par exemple :

```
function capturePhoto() {
  // Prend une photo en utilisant la fonction camera du téléphone
  navigator.camera.getPicture(onPhotoDataSuccess, onFail, {
    quality: 50,
    destinationType: destinationType.DATA_URL
  });
}
```

Pas de panique, ce code écrit en JavaScript n'aura bientôt plus beaucoup de secrets pour vous.

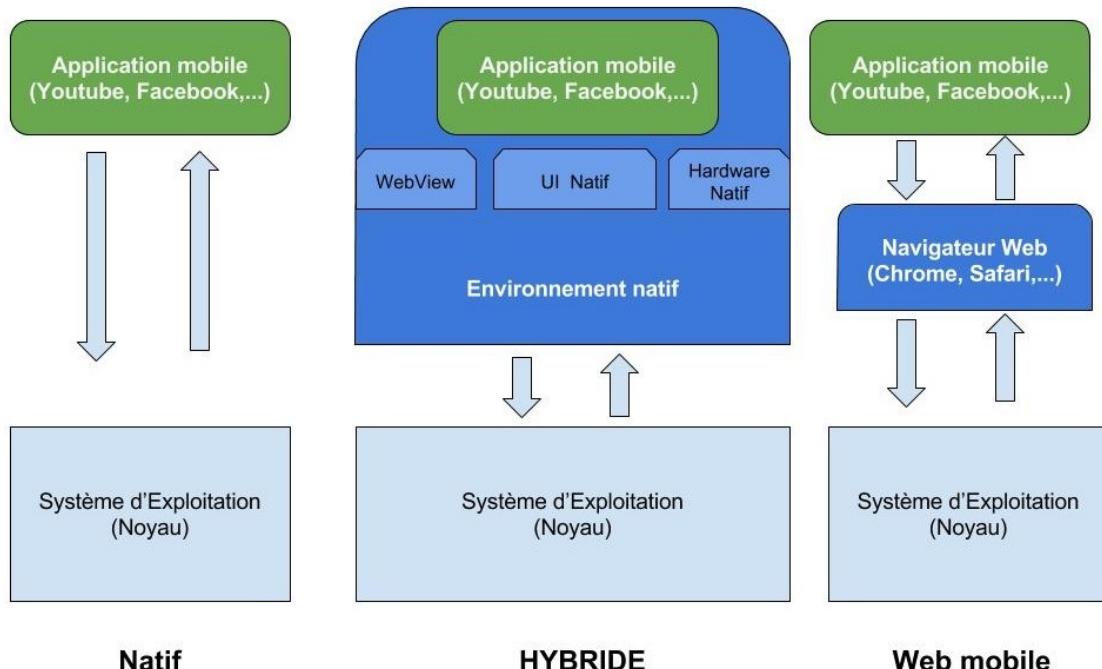
Pourquoi et quand faire le choix d'un développement Hybride

Une application hybride est avant tout une application native. Elle est téléchargée à partir d'un app store ou d'une tout autre place de marché. Elle a les mêmes fonctionnalités natives et à peu près les mêmes performances que toute application construite avec un SDK natif.

Une application hybride s'exécute dans un genre de navigateur en plein écran, appelé **WebView**, invisible pour l'utilisateur. Grâce à des plugins natifs personnalisables, elle peut accéder aux fonctionnalités natives de périphériques mobiles spécifiques (caméra, contacts, agenda,...), sans que le code principal ne soit lié à cet appareil.

Cela signifie que les applications hybrides peuvent fonctionner sur n'importe quelle plate-forme ou périphérique, tout cela à partir d'une base de code unique, tout en offrant un aspect et une convivialité natifs.

Le schéma en couche ci-dessous permet une meilleure comparaison de ce qu'une application hybride est vis-à-vis du natif et du web mobile :



Avant de vous engager dans un développement mobile, il est important de bien évaluer les avantages et les inconvénients de chacunes de ces méthodes. Voici quelques questions à se poser pour vous aider à vous lancer :

- Quelles plates-formes mobiles souhaitez-vous cibler ? Android ? iOS ? Windows Phone ?
- Voulez-vous distribuer votre application via un app Store ?
- Cherchez-vous à utiliser une ou plusieurs fonctionnalités clés de l'appareil mobile ?
- Quelles sont vos capacités techniques et/ou celle de votre équipe de développement ?

N'hésitez pas à noter ces questions, à y répondre, puis à les poser aux personnes pour ou avec qui vous prévoyez de travailler.

Voilà plus en détail ce que ces différentes questions peuvent impliquer.

Quelles plates-formes mobiles souhaitez-vous cibler ?

Si vous souhaitez cibler plus d'une plate-forme mobile, vous avez alors plusieurs choix possibles. Cela va de soi.

Et de toute évidence, le web mobile offre la solution la plus attrayante, surtout si vous possédez déjà une version web de ce que vous souhaitez développer pour les mobiles. Dans ce cas, un simple développement d'une version responsive de votre site peut être suffisant.

Mais, un développement hybride peut être aussi une excellente alternative.

Si ce que vous souhaitez proposer ne requiert pas d'avoir de très grosses performances logiciels, alors le natif n'est sûrement pas une bonne idée. Cependant, si vous ne voyez aucun inconvénient à recruter 3 développeurs différents ou à exploiter celui que vous avez déjà (si ce n'est vous), le priver de vacances jusqu'à ce que mort s'en suive, alors oui partez sur du natif. Bon j'exagère un peu, mais c'est à peu près ça.

Préparez-vous de toute façon, si votre choix se porte sur le natif, à parler couramment l'Objective-C ou le Swift pour iOS, Java ou Kotlin pour Android, et C-Sharp (C #) pour Windows Phone, pour ne citer que ces trois plateformes.

Voulez-vous distribuer votre application via un app Store ?

Si vous souhaitez distribuer votre application via une boutique d'applications mobiles, vous devrez alors créer soit une application hybride, soit une native. Vous aurez dans tous les cas besoin d'un site internet qui servira de plateforme à vos utilisateurs au cas où ils rencontreraient des problèmes avec votre application mobile.

Simon, là aussi une application web mobile ou [PWA](#) devrait suffire.

Vous cherchez à utiliser des fonctionnalités clés de l'appareil mobile?

Grâce aux PWA (Progressive Web Apps), on peut faire énormément de choses depuis un site mobile et utiliser des fonctionnalités qui dans le passé n'étaient accessibles qu'en développement natif ou hybride. Je consacre un chapitre entier à cette technologie à la [fin de ce livre](#).

Mais si les fonctions que vous ciblez dépassent le cadre du PWA, là aussi pas le choix, vous devrez passer par l'Hybride ou le natif. Et si vous souhaitez être irréprochable niveau interface utilisateur et performance, alors le natif devra être votre premier choix.

Quelles sont vos capacités techniques et/ou celle de votre équipe de développement ?

Grande question encore et loin d'être la moins importante. Si vous avez une grosse équipe de développement, vous êtes alors à l'abri de nombreux soucis. Vous aurez le choix d'utiliser l'une ou autre des trois méthodes précédentes. Vous êtes riche et vous le valez bien.

Vous aurez besoin, pour créer une application native, de développeurs expérimentés, maîtrisant à la fois les SDK (outils de développement) et les langages de programmation de chaque plate-forme que vous souhaitez cibler.

Mais si vous souhaitez ne pas mettre à profit autant de compétences techniques pour un minuscule projet qui ne vous rapportera pas plus de 1% de votre chiffre d'affaires, alors cela vaudrait peut être la peine de réfléchir à l'option Hybride qui pourrait vous faire économiser énormément de temps et d'argent. Payez ensuite des vacances à vos collaborateurs avec l'argent économisé via une prime d'intéressement ou exceptionnelle par exemple.

Dans la plupart des cas, développer en mode hybride offre le meilleur rapport qualité/prix, surtout si vous démarrez une première version de votre application. Ça tombe bien, Ionic permet la création d'applications hybrides et ce livre est là pour vous aider à décoller de la meilleure façon qui soit.

Pourquoi choisir Ionic ?

Open Source et 100% gratuit

Un avantage non considérable, surtout si l'on souhaite customiser un peu son travail, l'adapter à son contexte professionnel,...Gratuit, mais pas bradé pour autant. En effet, l'inconvénient d'un outil Open Source à parfois été son absence d'évolution et d'adaptation aux défis technologiques toujours plus important dans le temps. Ce n'est pas le cas de Ionic, qui est par exemple passé d'une version 1 déjà révolutionnaire, à une version 3 extrêmement riche. Au moment de la rédaction de ce livre, une version 4 (en phase alpha) s'apprête à être lancé publiquement.

Une large communauté

Qui n'a jamais connu la frustration de ne pas recevoir de réponse à sa question sur un forum destiné à vous accompagner dans l'utilisation de votre logiciel préféré ? Avec Ionic, il y a quasiment peu de chance que cela vous arrive. En plus d'une documentation déjà très riche et simple à prendre en main, le Framework fédère une très large communauté de développeurs enthousiastes près à vous aider en cas de pépin.

Rien que sur Stackoverflow en ce moment, le tag "[ionic framework](#)" est associé à plus de 31700 questions, c'est quasiment autant de réponses données sur ce site d'entraide.

Fondations solides

Ionic c'est d'abord Apache Cordova et ses nombreux plugins natifs, Angular de Google, NodeJS et bien d'autres technologies Open Source qui ont fait leur preuve depuis des années et ne cessent de croître à vitesse grand V.



Coder une fois, déployer partout

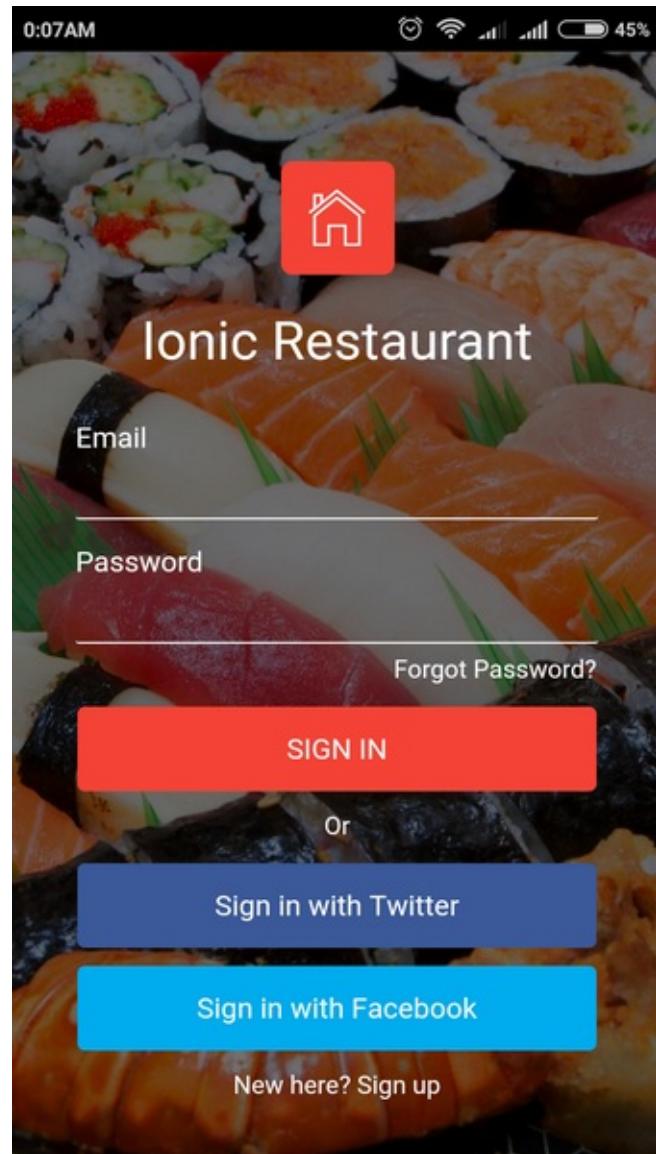
Et oui, c'est sûrement l'un des plus gros avantages de Ionic : la possibilité de développer votre application une fois, et la déployer sur plusieurs terminaux mobiles.

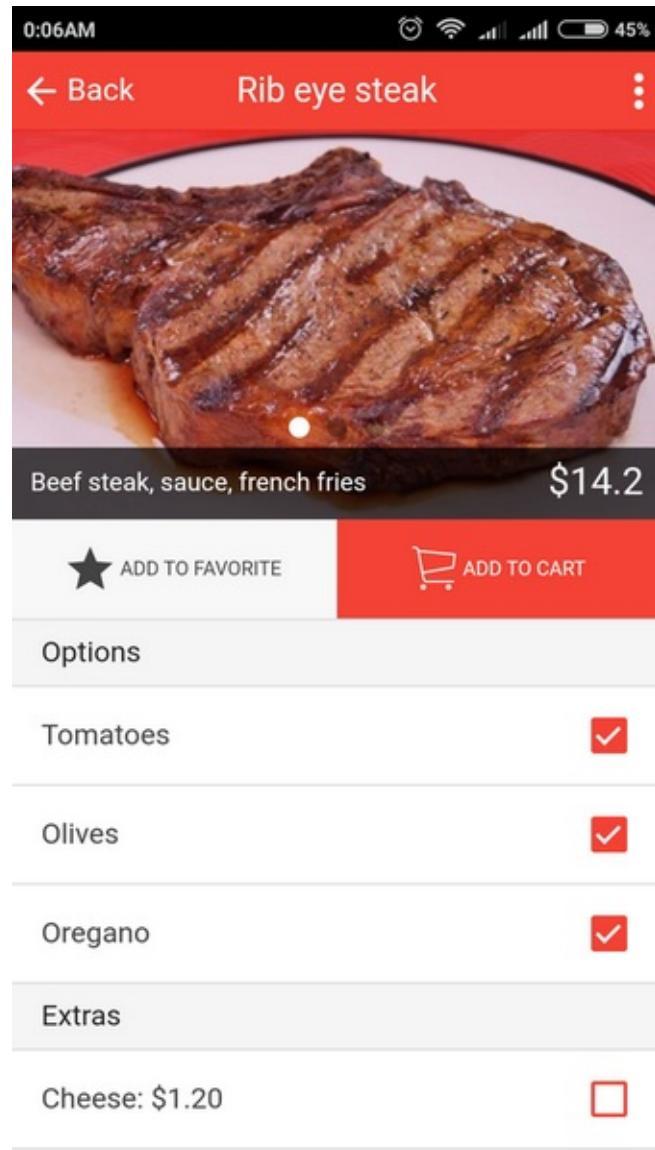


Alors que la plupart des outils de développement d'applications mobiles hybrides sont difficiles d'accès, parfois peu ou mal documentés, Ionic propose une architecture beaucoup plus simple à utiliser et une documentation claire et concise.

Des composants logiciels élégants

Le framework propose de nombreux composants et templates qui vous faciliteront énormément la vie. Pas besoin d'être un expert UX pour commencer à créer des applications mobiles élégantes et ergonomiques : quelques clics et quelques lignes de codes suffisent.





Tests et débogage simples

Tester une application mobile n'aura jamais été aussi simple. Ionic vous laisse le choix de tester votre application mobile soit directement depuis votre navigateur préféré, depuis un emulateur, votre téléphone connecté en USB ou alors via les applications [Ionic App Dev](#) et [Ionic View](#) (qui vous permet en autre de faire tester votre application à distance sans passer par un app store).

Un écosystème riche

Une autre force de Ionic est très certainement son large panel de produits à la disposition des développeurs. On peut citer par exemple :

- **Les outils de débogages** : Ionic view et Ionic Dev app, comme vu précédemment.
- **Ionic Cloud** : pour l'hébergement de son code (à la github), la compilation, la publication sur les app stores et le monitoring (bug, logs, statistiques,...)
- **Ionic creator** : pour créer une application sans saisir une seule ligne de code. Pratique quand on débute.
- **Ionic Market** : pour trouver l'inspiration grâce à des templates, des kits, des plugins,... gratuits et prêts à l'emploi. Vous pouvez aussi utiliser cette plateforme pour mettre à la vente vos propres réalisations

¹. John Ronald Reuel Tolkien, plus connu sous la forme J. R. R. Tolkien, est un écrivain, poète, philologue, essayiste et professeur d'université anglais. Il est principalement connu pour ses romans *Le Hobbit* et *Le Seigneur des anneaux*. (Sources : [wikipédia](#)). ↪

Chap 2 - Cas pratique : DuckCoin, la cryptomonnaie sur mobile

Explorer Ionic et son écosystème peut vite devenir long, alors pour faire à peu près le tour du sujet, je vous propose dans ce livre de partir d'un projet concret qui va nous permettre d'aborder tous les concepts techniques dont vous aurez besoin pour lancer votre propre application mobile.

Après réflexion, et aimant surfer sur les tendances technologiques du moment, je vous propose d'explorer l'univers des cryptomonnaies en développant notre propre monnaie numérique, basée sur une blockchain du même nom et que nous appellerons **DuckCoin**, en hommage à la mascotte du département MMI de l'IUT de Troyes où j'ai le plaisir d'intervenir.

Un site internet a été créé pour l'occasion. Vous pourrez le visiter, vous inscrire et tester notre cryptomonnaie solidaire : <https://duckcoin.charlesen.fr>

De plus, le code source de ce projet est librement disponible à l'adresse : <https://github.com/charlesen/duckcoin>.

Cryptomonnaies et blockchain

C'est à la une de quasiment tous les journaux d'information. Par une seule semaine sans que l'on vous parle ci et là du Bitcoin, de Ripple ou d'une autre crypto-monnaie en vogue.

Une crypto-monnaie est selon Wikipédia :

Une monnaie virtuelle utilisable sur un réseau informatique décentralisé, de pair à pair

Dit autrement, c'est comme remplacer ses euros, ses dollars ou ses francs CFA par une devise stockée en ligne ou directement dans votre ordinateur ou clé usb. Bitcoin, la plus célèbre des crypto-monnaies et celle qui vaut le plus chère, a été lancé quelques temps seulement après la crise financière de 2008 : la fameuse crise des subprimes.

Quelques années plus tard, on dénombre plus de 1000 crypto-monnaies selon le site internet [CoinMarketCap](#) pour une capitalisation boursière (valeur au prix du marché de l'ensemble des crypto-monnaies en circulation) de plus de 740 milliards de dollars. C'est juste énorme et ce n'est rien comparé au potentiel de ce nouveau marché.

Principales crypto-monnaies

Bitcoin

Bitcoin est une devise virtuelle pair-à-pair décentralisée qui fonctionne grâce à ses utilisateurs, sans autorité centrale ni intermédiaire. Elle vaut à l'heure où j'écris ces quelques lignes autour de 6 616,51 dollars, soit environ 5 405,71 euros, soit plus de 3,5 millions de Franc CFA. C'est juste énorme.

1 Bitcoin = €5 405,71

Ripple

Ripple est un système de règlement brut en temps réel, un marché des changes et un réseau de transfert de fonds. Également appelé le Ripple Transaction Protocol ou Protocole Ripple, il est construit sur un protocole Internet distribué et open source, un registre de consensus et une monnaie native appelée XRP. Lancé en 2012, le réseau Ripple a pour objectif de permettre des transactions financières mondiales sécurisées, instantanées et presque gratuites, de toute taille sans rejets de débit.

Ripple et Bitcoin utilisent une méthode différente pour parvenir à un consensus réseau. Ripple utilise un processus de consensus itératif, tandis que Bitcoin utilise le « Proof of Work » (minage). Par conséquent, Ripple est plus rapide que Bitcoin. La finalisation des transactions ne prend que quelques secondes.

1 Ripple = €0,382906

Ethereum

Ethereum est une devise virtuelle développée sur une plateforme logicielle ouverte basée sur la technologie blockchain qui permet aux développeurs de créer et déployer des applications décentralisées qui exécutent des contrats intelligents (« **smart contracts** »). C'est fin 2013 que Vitalik Buterin, un chercheur et développeur en crypto-monnaie, propose Ethereum.

À l'instar de Bitcoin, Ethereum est un réseau de blockchain publique. Bien qu'il existe des différences techniques considérables entre les deux, la distinction la plus importante est que Bitcoin et Ethereum diffèrent considérablement en termes d'objectifs et de capacités. Tandis que la blockchain bitcoin est utilisée pour suivre la propriété d'une devise virtuelle (Bitcoins), la blockchain Ethereum se concentre sur l'exécution du code de programmation de toute application décentralisée.

1 Ethereum = €300,78

La Blockchain pour les nuls

Selon Wikipédia :

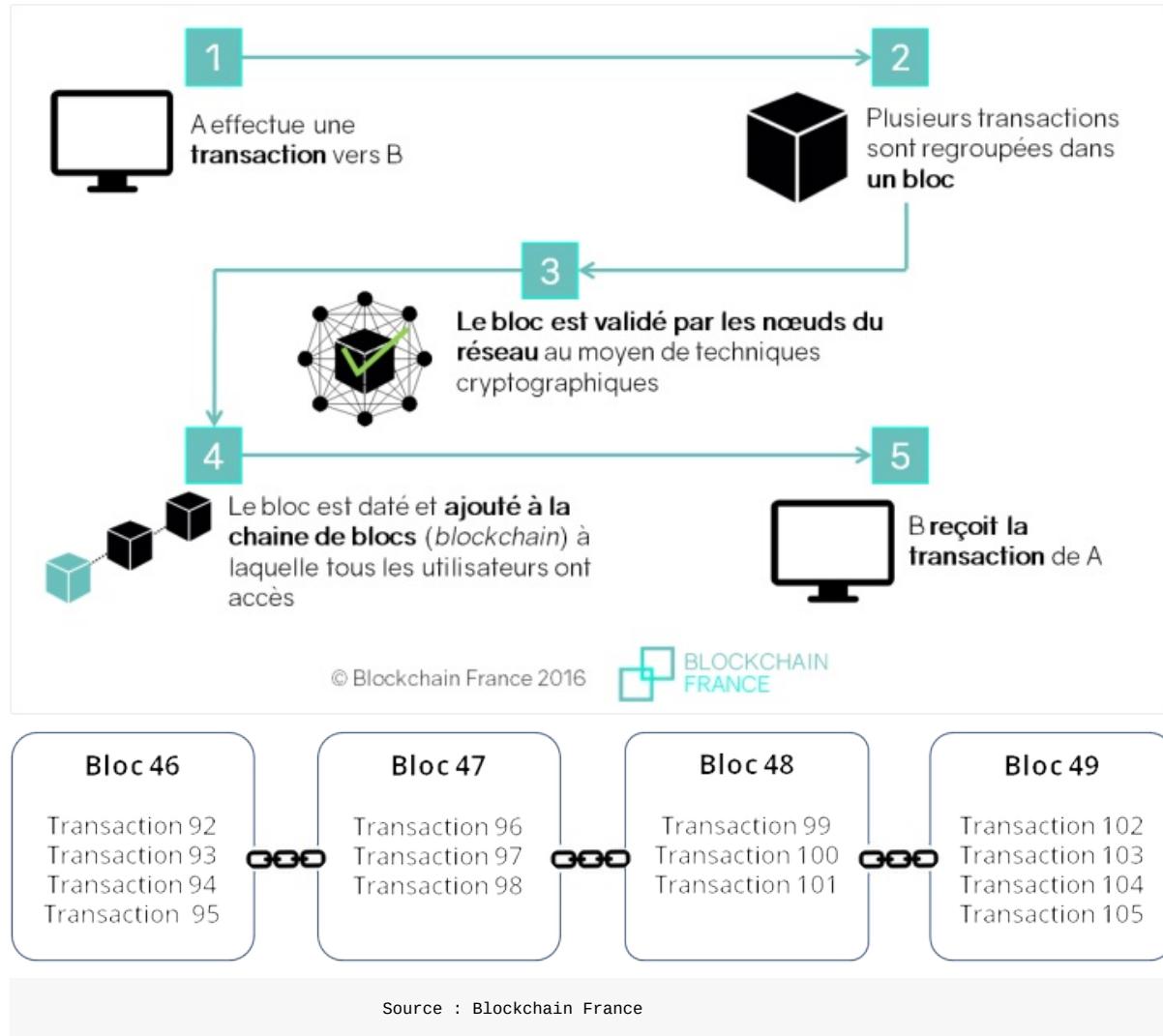
Une blockchain, ou chaîne de blocs, est une technologie de stockage et de transmission d'informations sans organe de contrôle. Techniquement, il s'agit d'une base de données distribuée dont les informations envoyées par les utilisateurs et les liens internes à la base sont vérifiés et groupés à intervalles de temps réguliers en blocs, l'ensemble étant sécurisé par cryptographie, et formant ainsi une chaîne

Une blockchain contient des données créées par différents utilisateurs dans le temps et chaque acteur ou noeud de la blockchain possède une copie de celle-ci.

Il existe différents types de blockchains : celles qui sont privés, c'est à dire qu'un nombre limité d'acteurs, choisi arbitrairement, a le droit d'agir sur la blockchain, et celles qui sont publiques et donc accessibles à tout le monde sans aucune restriction. Comme l'a dit le mathématicien Jean-Paul Delahaye, une blockchain publique comme celle de Bitcoin peut être assimilée à

un très grand cahier, que tout le monde peut lire librement et gratuitement, sur lequel tout le monde peut écrire, mais qui est impossible à effacer et indestructible.

Une image valant mieux qu'un discours, voici comment on pourrait représenter la technologie Blockchain de manière simple :



Les applications de la Blockchain sont multiples car elle permet d'éliminer les tiers de confiance habituels que sont les banques, les assurances, les notaires,...et même l'Etat.

Si l'euro a de la valeur aujourd'hui c'est d'abord parce que cette valeur nous la lui accordons et que les banques et les Etats européens veillent à ce que cela soit toujours le cas en limitant sa quantité et sa provenance (BCE), en punissant pénallement la création de fausses monnaies,...

Avec la blockchain, ces tiers de confiance peuvent s'en aller en paix, la confiance étant répartie entre les différents acteurs du réseau blockchain. Rien ne nous empêche alors de créer notre propre monnaie, la distribuer et de la valoriser auprès d'un large panel de développeurs d'applications mobiles enthousiastes. C'est ce que nous allons faire tout au long des chapitres avec la cryptomonnaie **DuckCoin**.

Duckcoin : principes de fonctionnement

DuckCoin sera comme nous l'avons déjà dit une crypto-monnaie basée sur une blockchain publique. L'application mobile que nous allons développer tout au long de ce livre portera le même nom.

De plus, la monnaie fonctionnera à peu près sur le même principe que le Bitcoin, en tout cas en ce qui concerne son architecture.

Ce livre n'étant pas consacré à la Blockchain et aux cryptomonnaies, vous n'êtes pas obligé de lire la suite de ce chapitre et pouvez directement passer à la suite. Mais si vous êtes un tant soit peu curieux, alors restez, vous ne serez pas dessus je pense.

Construction de la Blockchain

Pour développer la blockchain sur laquelle sera construite notre cryptomonnaie, nous allons utiliser le langage de programmation Python, qui est assez simple à maîtriser.

```
class Blockchain(object):
    def __init__(self):
        # Constructeur : on initialise ici la chaîne et la liste qui contiendra les transactions
        self.chain = []
        self.current_transactions = []

    def new_block(self):
        # Permet la création d'un nouveau block qui sera à la chaîne de blocs
        pass

    def new_transaction(self):
        # Ajoute une nouvelle transaction à la liste des transactions.
        # Un bloc peut contenir plusieurs transactions, toutes les unes que les autres
        pass

    @staticmethod
    def hash(block):
        # Permet le hashage d'un Bloc
        pass

    @property
    def last_block(self):
        # Renvoie le dernier bloc de la chaîne
        pass
```

La classe Blockchain sera responsable de la gestion de la chaîne. Elle va stocker les transactions et pourra grâce à des méthodes ajouter de nouvelles transactions ou de nouveaux blocs à la chaîne.

Un Bloc possède un **index**, c'est à dire un numéro qui permet de le situer dans la chaîne, un système d'horodatage appelé **timestamp**¹, une **liste de transactions**, une **preuve de travail** ou proof of work en anglais que nous abrégerons *pow*, un **hash**, c'est à dire une empreinte numérique rendant le bloc unique en son genre, le **hash** du bloc précédent (*previous_hash*). Prenons par exemple le premier bloc de notre blockchain. Il ressemble à ceci :

```
block = {
    'index': 1,
    'timestamp': 1506057130.100625,
    'transactions': [
        {
            'sender': "2327147fe1f5426f9dd545de4b27ee00",
            'recipient': "82dec7f5cdafa2934df3954a5c7c7da5df1f",
            'amount': 30,
        }
    ],
    'proof': 952,
    'hash': "e24db68eb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824",
    'previous_hash': ""
}
```

Ici, le premier bloc ne possède pas de hash pour le bloc précédent (*previous_hash*), ce qui est normal. Par contre, les blocs suivants devront l'avoir obligatoirement. C'est d'ailleurs cela qui garantit en partie la fiabilité de la blockchain. En effet, si un seul bloc devait être falsifié (un utilisateur qui mentirait sur le montant total de ses avoirs en modifiant un bloc), c'est toute la chaîne qui deviendrait invalide.

A présent rentrons plus en détail sur le fonctionnement des méthodes de la classe BlockChain.

Gestions des transactions

La méthode **new_transaction** est en charge de l'ajout de nouvelles transactions au sein d'un bloc.

```
class Blockchain(object):
    ...

    def new_transaction(self, sender, recipient, amount):
        """
        Création d'une nouvelle transaction qui sera intégré au dernier bloc à inclure dans la blockchain
        :param sender: <str> Adresse (hash) du destinataire
        :param recipient: <str> Adresse (hash) du destinataire
        :param amount: <int> Montant envoyé par le 'sender' au 'recipient'
        :return: <int> index du bloc qui stockera cette transaction
        """

        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1
```

Chaque fois qu'un utilisateur souhaitera envoyer de l'argent à un autre, c'est cette méthode qui sera appelée. Le tout (la transaction) sera stockée dans le prochain bloc à miner (notion que nous aborderons plus loin).

Gestion des blocs

A la création de la blockchain, nous allons devoir créer un bloc initial, qui stockera les toutes premières transactions de la blockchain. C'est à l'intérieur de ce bloc par exemple que l'on pourra stocker les transactions permettant d'envoyer de l'argent aux 20 premiers utilisateurs de notre cryptomonnaie. Et oui, avoir de l'argent en stock c'est bien, mais si personne ne s'en sert, elle n'a aucune valeur.

Ce bloc initial est appelé dans l'univers des cryptomonnaies, le bloc **Genesis**, pour genèse en français. Et avant la genèse, il n'y a rien...en principe.

```
import json
from hashlib import sha256

from time import time

class Blockchain(object):
    def __init__(self):
        self.current_transactions = []
        self.chain = []

        # Création du bloc initial genesis
        self.new_block(previous_hash='', proof=100)

    def new_block(self, proof, previous_hash=None):
        """
        Création d'un nouveau bloc dans la Blockchain
```

```

:param proof: <int> valeur retourné l'algorithme de preuve de travail
:param previous_hash: (Optionnel pour le premier bloc) <str> Hash du bloc préc.
:return: <dict> Nouveau Bloc
"""

block = {
    'index': len(self.chain) + 1,
    'timestamp': time(),
    'transactions': self.current_transactions,
    'proof': proof,
    'previous_hash': previous_hash or self.hash(self.chain[-1]),
}
# Remise à zéro de la liste des transactions
self.current_transactions = []

self.chain.append(block)
return block

def new_transaction(self, sender, recipient, amount):
    """
    Création d'une nouvelle transaction qui sera intégré au dernier bloc à inclure dans la blockchain
    :param sender: <str> Adresse (hash) du destinataire
    :param recipient: <str> Adresse (hash) du destinataire
    :param amount: <int> Montant envoyé par le 'sender' au 'recipient'
    :return: <int> index du bloc qui stockera cette transaction
    """
    self.current_transactions.append({
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
    })
    return self.last_block['index'] + 1

@property
def last_block(self):
    return self.chain[-1]

@staticmethod
def hash(block):
    """
    Création d'un hash du bloc avec la fonction de hashage SHA-256
    :param block: <dict> Block
    :return: <str>
    """

    # On ordonne le bloc avant de le serialiser
    block_string = json.dumps(block, sort_keys=True).encode()
    return sha256(block_string).hexdigest()

```

La Preuve de travail, Proof of Work (pow)

La preuve de travail est la méthode qui permet de créer ou miner un nouveau bloc de la blockchain. Il s'agit d'un algorithme permettant, comme le rappel Wikipédia :

de dissuader, sur un réseau informatique, des attaques par déni de service et autres abus de service tels que le spam en requérant de la puissance de calcul et de traitement par ordinateur au demandeur de service. C'est un système difficile à produire car il est coûteux en temps et en énergie.

Lorsqu'un utilisateur effectue une transaction, celle-ci est, comme on l'a vu, stockée dans un bloc. Une fois que le bloc possède un certain nombre de transactions, il devra être intégré à la blockchain pour sauvegarde. C'est à ce moment là qu'intervient le minage par preuve de travail. Chaque noeud (ordinateur) va recevoir le bloc souhaitant intégrer la

chaine et effectuer un certain nombre de calculs suffisamment longs pour dissuader un noeud pirate. Et plus la chaîne est longue comme celle de Bitcoin et d'autres cryptomonnaies, et plus le calcul est énergivore et décourageante pour le noeud pirate.

Le noeud ayant réussi en premier à résoudre l'énigme se voit récompenser d'une somme en crypto. Bitcoin récompense par exemple ses mineurs à hauteur de 25 BTC (à l'heure actuelle).

Une caractéristique de la pow est l'asymétrie du coût de calcul : le travail doit être difficilement réalisable pour le demandeur, mais facilement vérifiable pour un tiers.

Prenons par exemple une preuve de travail bien connu, le Captcha, que l'on retrouve sur de nombreux sites internet.



Si pour un humain, saisir ces informations est déjà assez difficile, imaginez ce qu'éprouvera un robot spammeur.

Pour en savoir plus, n'hésitez pas à lire le bel article de **Cryptoeencyclopedia** :

<https://www.cryptoeencyclopedia.com/single-post/Quest-ce-que-le-consensus-Proof-of-Work->

Pour notre algorithme de preuve de travail, choisissons arbitrairement que le hash d'un entier X, multiplié par un autre entier Y devra absolument se commencer par 3 chiffres 0. On aurait ceci par exemple :

```
hash(x * y) = 000ecad23dc...
```

Cela donnerait le code suivant :

```
import json

from hashlib import sha256
from time import time
from uuid import uuid4


class Blockchain(object):
    ...

    def proof_of_work(self, last_proof):
        """
        Un algorithme de Preuve de travail :
        - Trouver un nombre x tel que hash(xy) commence par 3 zeros
        - x étant la preuve précédente (last_proof)
        - y étant une valeur que l'on incrémentera en partant de zero et qui
          à la fin du calcul deviendra la nouvelle preuve (proof)
        :param last_proof: <int>
        :return: <int>
        """
        proof = 0
        while not self.valid_proof(last_proof, proof):
            proof += 1

        return proof
```

```

@staticmethod
def valid_proof(last_proof, proof):
    """
    Validation du résultat de la preuve de travail : est-ce que hash(last_proof, proof)
    commence bien par 3 zéros ?
    :param last_proof: <int> Preuve précédente
    :param proof: <int> Preuve actuelle
    :return: <bool>
    """
    proof_hash = sha256(str(last_proof*proof)).hexdigest()
    return proof_hash[:3] == "000"

```

On aurait pu compliquer le calcul en testant un nombre de zéros beaucoup plus grand, mais l'on se contentera de 3 zéros. Si vous passez de 3 à 4, le même calcul avec les mêmes valeurs en entrée est largement plus long. Donc on imagine bien que plus l'on augmente le nombre de zéros, et plus les choses se compliquent.

Un test avec une valeur de x=2018, permet d'avoir un résultat en 0.07s environ

```

import time

x = 2018 # Dernière preuve
y = 0 # Valeur que l'on incrémentera jusqu'à trouver le bon résultat

debut = time.time()
while not valid_proof(x,y):
    y += 1
    print y

fin = time.time()
duree = fin-debut

# Hash trouvé : 000e7ea9705df1fe65fe077d5054fe4a12aa6bbe074d5060ed9f0b251e16d0f9
# La solution est y = 566 au bout de 0.0716059207916 s

```

Le même test prend plus de 6s pour un nombre de zéros égal à 4.

Pour rappel, la preuve de travail du Bitcoin, qui ressemble à peu près à celle que l'on a implementé, utilise un nombre de zéros égal à 18 (voir image ci-dessous). C'est quand même énorme !

Bloc #517043

Sommaire		Hashes	
Nombre de transactions	8	Hash	00000000000000001bfaabb992b104e81c971d0ea394a0b357e637dea8c034
Somme des outputs	16.42463078 BTC	Bloc précédent	000000000000000015276a6b8f5478950f226c4ec97dbb5928ff574985d508
Volume estimé des transactions	0.13861477 BTC	Bloc(s) suivant(s)	
Frais des transactions	0.00119436 BTC	Merkle Root	c06e8990f81c45774cb34451b746c91980846978734ff389ee919f9dd651566
Hauteur	517043 (Chaîne principale)		
Date (timestamp)	2018-04-07 09:18:45		
Date de réception	2018-04-07 09:18:45		
Relayé par	Unknown		
Difficulté	3,511,060,552,899.72		
Morceaux	391129783		
Taille	2.915 kB		
Poids	10.0 kWU		
Version	0x20000000		

Interaction avec la Blockchain DuckCoin

Pour l'affichage et les interactions avec la blockchain, nous utiliserons un framework Python nommé Flask, robuste et très simple à prendre en main. Si vous souhaitez en savoir plus sur Flask, n'hésitez pas à lire l'excellente documentation en ligne : <http://flask.pocoo.org/>

Notre cryptomonnaie qui est disponible à l'adresse : <https://duckcoin.charlesen.fr/> peut être exploitée via les actions suivantes :

Action	Description
/transactions/new	Permet d'ajouter une nouvelle transaction
/mine	Minage d'un nouveau bloc
/chain	Retourne la blockchain complète

Maintenant que notre outil est en place, nous pouvons seirenement nous lancer dans la création de l'application mobile qui nous permettra d'interagir avec la Blockchain, gérer notre portefeuille, envoyer ou recevoir de l'argent sous forme de tokens **DCK**.

¹. Le timestamp (unix) désigne le nombre de secondes écoulées depuis le 1er janvier 1970 à minuit UTC précise



Chap 3 - Installation de Ionic et première prise en main

Ionic utilise un certain nombre d'outils permettant de créer rapidement une application mobile. Pris séparément, ils sont plus ou moins efficace, voir indépendant, mais mis en ensemble, ils sont d'une redoutable efficacité.

Parmi ces outils nous pouvons citer principalement :

- **Ionic CLI** : c'est le couteau suisse de Ionic, un ensemble de fonction disponible en ligne de commandes pour créer une application, la compiler, la déployer,...
- **Apache Cordova** : un framework open-source développé par la Fondation Apache. Il permet de créer des applications pour différentes plateformes en HTML, CSS et JavaScript.
- **NodeJS** : est un logiciel permettant de développer et d'exécuter du code JavaScript côté serveur, contrairement à ce qu'on a l'habitude de voir avec le javascript côté client.
- **NPM** : le gestionnaire de paquet de NodeJS
- **Angular** : un framework Javascript développé par Google
- **TypeScript** : un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript.
- **SASS** : un langage de génération de feuilles de style (CSS dynamique)
- ...

NodeJS et NPM

NodeJS en bref



Depuis son commencement, JavaScript, a été, comme vous le savez très certainement, est un langage dit côté client. Mais les choses ont quelque peu évolué avec NodeJS : cette technologie permet en effet d'executer du code écrit en JavaScript, aussi bien sur un navigateur (côté client), que côté serveur, tout comme des langages comme le Python ou encore le PHP.

De plus, NodeJS, à l'instar de Ionic, est Open Source, gratuit et disponible pour différentes plateformes (Windows, Linux, Unix, Mac OS,...)

NPM : Node Package Manager



Comme son nom peut le suggérer, NPM est le gestionnaire de packet de NodeJS, qui étant très modulaire, voit son ecosystème constamment enrichi par des modules développés par les membres de sa large communauté.

Installation

Windows et Mac OS

Pour installer NodeJS, il suffit simplement d'aller à l'adresse : <https://nodejs.org/en/download/>, de télécharger le gestionnaire d'installation au format **.msi** pour windows et **.pkg** pour Mac OS. Laissez-vous ensuite simplement guider. Le gestionnaire installera également NPM.



Downloads

Latest LTS Version: 8.11.1 (includes npm 5.6.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

32-bit	64-bit	
32-bit	64-bit	
	64-bit	
	64-bit	
32-bit	64-bit	
ARMv6	ARMv7	ARMv8

Ouvrez un invite de commande et saisissez :

```
node -v
```

Vous devriez voir s'afficher la version actuelle de NodeJs.

Linux (Ubuntu)

sous linux et en particulier Ubuntu, il vous suffit de saisir les commandes suivantes depuis un invite de commandes :

```
$ sudo apt-get update
$ sudo apt-get install nodejs npm
```

une fois l'installation effective, il faut encore créer les liens symboliques suivants :

```
$ sudo ln -s /usr/bin/nodejs /usr/local/bin/node
$ sudo ln -s /usr/bin/npm /usr/local/bin/npm
```

Pour vérifier que tout s'est bien passé, il vous suffit de saisir la commande suivante qui vous retournera la version actuelle de Node :

```
$ node -v
v8.11.1
```

Ionic CLI et Cordova

Une fois Node et NPM installés, le reste se passera en ligne de commandes. Ouvrez donc votre terminal préféré et saisissez les commandes suivantes pour installer Ionic et Cordova :

```
$ sudo npm install -g ionic cordova
```

Le paramètre **"-g"** permet une installation globale de ces outils. De cette manière, vous n'aurez pas besoin d'être dans un répertoire particulier pour utiliser les commandes **ionic** ou **cordova**, sauf pour des actions comme la compilation qui requiert d'être à l'intérieur d'un projet.

Ce paramètre implique aussi que vous devrez lancer les commandes précédentes en tant qu'Admin sous Windows (clic-droit, démarrer l'invite de commande en tant qu'administrateur) et que sous Linux, vous êtes obligé d'utiliser le **"sudo"**. Il est bien sûr possible de se passer d'un **"sudo"** en bricolant un peu sa config npm², mais ceci dépasse l'objet de ce livre.

Avant d'aller plus loin, il sera nécessaire d'installer d'autres logiciels comme le SDK de Java ou celui d'Android. Si ces logiciels sont déjà installés, vous pourrez directement passer à la suite, sinon, suivez le guide.

Autres utilitaires

Java SDK

Windows et Mac OS

Pour installer le SDK de Java sous Windows et Mac, il vous suffit de visiter le site :

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html> et de choisir le fichier adapté à votre machine (32 ou 64 bits)

Linux

Mise à jour des dépôts

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
```

Installation d'OpenJDK

```
sudo apt-get install openjdk-8-jdk
```

Android SDK

La meilleure façon d'installer le SDK d'Android est encore d'installer Android Studio. Pour ce faire, rien de plus simple, il suffit de visiter le site <https://developer.android.com/studio/index.html#downloads> et de télécharger le paquet associé à votre OS.

Windows

Une fois le téléchargement effectué, vous n'aurez plus qu'à lancer l'installation en cliquant sur le fichier au format **.exe** et suivre le setup. L'installation du SDK se fera en même temps.

Voilà, c'est tout.

Mac OS

1. Lancer l'installation en cliquant sur le fichier au format **.dmg** téléchargé précédemment.
2. Glisser-déposer (Drag-n-drop) ensuite Android Studio dans le dossier Applications
3. Le setup devrait ensuite finaliser l'installation du SDK

Voilà.

Linux

1. Décompresser le fichier **.zip** téléchargé précédemment dans un dossier approprié. Je vous propose le dossier **/opt/** de manière à le partager entre les différents acteurs de votre OS.
2. Ouvrez un invite de commandes (CTRL + ALT + T) et exécuter le fichier **/opt/android-studio/bin/studio.sh**.
3. Suivez le setup

Si votre OS est une machine 64-bit, vous allez devoir installer quelques dépendances logicielles :

```
$ sudo apt-get install libcurl4-openssl-dev libncurses5-dev libstdc++6 lib32z1 libbz2-1.0
```

Xcode et ios-sim (Mac OS uniquement)

Pour installer **Xcode**, il vous suffit de visiter l'URL suivante <https://developer.apple.com/xcode/> et de cliquer sur **"Download"**.

Pour l'émulateur **ios-sim**, depuis votre terminal, saisissez la commande suivante :

```
$ sudo npm install -g ios-sim
$ ios-sim -version # Pour vérifier que tout s'est bien passé
```

Git

Ionic utilise le gestionnaire de dépôt Git dans son workflow de développement actuel. Pour l'installer, rien de plus simple, il vous suffit d'aller à la page de téléchargement suivante : <https://git-scm.com/downloads> et choisir le paquet correspondant à votre OS.

Sous Linux, il est également possible de l'installer en saisissant simplement la commande :

```
$ apt install git
```

Installation via un script Bash (Ubuntu)

Un script trouvé sur [github](#)¹, et que j'ai un peu adapté, fait assez bien le travail, car il vous permet d'installer tous ces utilitaires via un seul fichier bash.

```
#!/bin/bash
# Ubuntu Developer Script For Ionic Framework
# Created by Nic Raboy
# http://www.nraboy.com
#
#
# Télécharge et configure les logiciels suivants :
#
# Java JDK
# Apache Ant
# Android
# Apache Cordova
# Ionic Framework
# Gradle
```

```

HOME_PATH=$(cd ~/ && pwd)
INSTALL_PATH=/opt
ANDROID_SDK_PATH=/opt/android-sdk
NODE_PATH=/opt/node
GRADLE_PATH=/opt/gradle

# x86_64 ou i686
LINUX_ARCH=$(lscpu | grep 'Architecture' | awk -F\: '{ print $2 }' | tr -d ' ')

# Android Linux SDK pour les architectures x64 et x86
ANDROID_SDK_X64="http://dl.google.com/android/android-sdk_r24.4.1-linux.tgz"
ANDROID_SDK_X86="http://dl.google.com/android/android-sdk_r24.4.1-linux.tgz"

# Gradle
GRADLE_ALL="https://services.gradle.org/distributions/gradle-2.9-all.zip"

if [ "$LINUX_ARCH" == "x86_64" ]; then
    # Add i386 architecture
    dpkg --add-architecture i386
fi

# Mise à jour des dépôts Ubuntu
apt-get update

cd ~/Desktop

if [ "$LINUX_ARCH" == "x86_64" ]; then

    wget -c "$ANDROID_SDK_X64" -O "android-sdk.tgz" --no-check-certificate
    wget -c "$GRADLE_ALL" -O "gradle.zip" --no-check-certificate

    tar zxvf "android-sdk.tgz" -C "$INSTALL_PATH"
    unzip "gradle.zip"
    mv "gradle-2.9" "$INSTALL_PATH"

    cd "$INSTALL_PATH" && mv "android-sdk-linux" "android-sdk"
    cd "$INSTALL_PATH" && mv "gradle-2.9" "gradle"

    # Dépendances pour les architecture x86
    apt-get install -qq -y libc6:i386 libgcc1:i386 libstdc++6:i386 libz1:i386

else

    wget -c "$ANDROID_SDK_X86" -O "android-sdk.tgz" --no-check-certificate
    wget -c "$GRADLE_ALL" -O "gradle.zip" --no-check-certificate

    tar zxvf "android-sdk.tgz" -C "$INSTALL_PATH"
    unzip "gradle.zip"
    mv "gradle-2.9" "$INSTALL_PATH"

    cd "$INSTALL_PATH" && mv "android-sdk-linux" "android-sdk"
    cd "$INSTALL_PATH" && mv "gradle-2.9" "gradle"

fi

cd "$INSTALL_PATH" && chown root:root "android-sdk" -R
cd "$INSTALL_PATH" && chmod 777 "android-sdk" -R

cd ~/

# MAJ du PATH de manière persistante
echo "export PATH=$PATH:$ANDROID_SDK_PATH/tools" >> ".profile"
echo "export PATH=$PATH:$ANDROID_SDK_PATH/platform-tools" >> ".profile"
echo "export PATH=$PATH:$GRADLE_PATH/bin" >> ".profile"

# MAJ du PATH de manière temporaire le temps de l'installation
export PATH=$PATH:$ANDROID_SDK_PATH/tools

```

```

export PATH=$PATH:$ANDROID_SDK_PATH/platform-tools
export PATH=$PATH:$GRADLE_PATH/bin

# Installation du JDK, Apache Ant et Git
apt-get -qq -y install default-jdk ant
apt-get install git

# MAJ de la variable d'environnement JAVA_HOME
export JAVA_HOME=$(find /usr -type l -name 'default-java')
if [ "$JAVA_HOME" != "" ]; then
    echo "export JAVA_HOME=$JAVA_HOME" >> ".profile"
fi

# Installation d'Apache Cordova et du framework Ionic
npm install -g cordova
npm install -g ionic

cd "$INSTALL_PATH" && chmod 777 "node" -R
cd "$INSTALL_PATH" && chmod 777 "gradle" -R

# Suppression des fichiers d'installation
cd ~/Desktop && rm "android-sdk.tgz"
cd ~/Desktop && rm "nodejs.tgz"
cd ~/Desktop && rm "gradle.zip"

echo "-----"
echo "Redémarrez votre session Ubuntu pour finaliser l'installation..."

```

Copier le code ci-dessus dans fichier `install_ionic.sh` puis executer le :

```

$ chmod u+x install_ionic.sh
$ ./install_ionic.sh

```

Redémarrez ensuite votre machine.

Création d'un compte Ionic PRO

Cette étape va nous permettre de disposer d'un compte sur le cloud de Ionic. On en parle en détails au [Chapitre 10](#), mais globalement, Ionic Cloud permet de :

- Compiler une application sans devoir installer sur son local toutes les dépendances nécessaires à cette compilation.
- Faire tester votre application en avant-première via l'application Ionic View
- Gérer les erreurs levées dans l'application mobile
- ...

La création d'un compte est gratuite et pour ce faire, rendez-vous à l'adresse <https://dashboard.ionicjs.com/signup> et remplissez le formulaire pour compléter votre inscription.



Sign up for Ionic Pro

Start shipping better apps

Name

Email

Username

Password

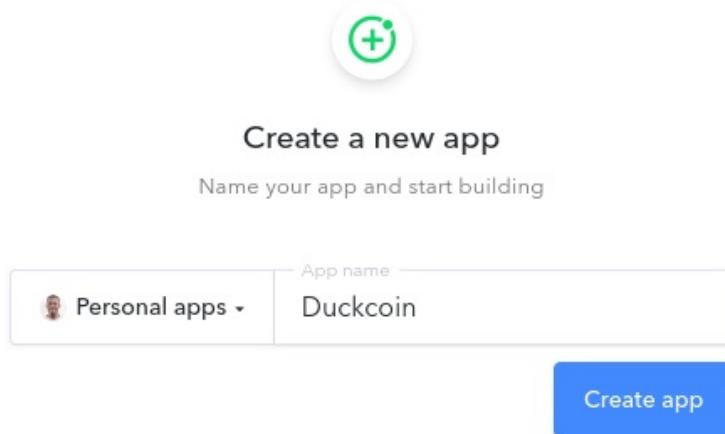
GET STARTED

By signing up you agree to our [Terms of Service](#) and [Privacy Policy](#)

Already have a Pro account? [Log in](#)

Connectez-vous ensuite à Ionic PRO et cliquez sur le bouton "**New app**" pour créer une application que l'on liera plus tard à notre application mobile.

Donnez un nom à cette nouvelle app. Par exemple "**Duckcoin**".



Une fois l'application créée, un identifiant unique lui est attribuée. Cet identifiant nous sera utile à la création de notre application depuis notre poste de travail.

Voilà, vous y êtes. On va donc pouvoir créer notre première application mobile.

Première application Ionic

Pour créer votre première application, rien de plus simple :

```
$ ionic start monAppli tabs
✓ Creating directory ./monAppli - done!
✓ Downloading and extracting tabs starter - done!

? Would you like to integrate your new app with Cordova to target native iOS and Android? (y/N)
```

A la question "*Would you like to integrate your new app with Cordova to target native iOS and Android?*" saisir "**y**".

Et à la question "*Install the free Ionic Pro SDK and connect your app?*", répondez aussi par un "**y**".

Vous allez devoir entrez vos identifiants et générer une paire clé privé/publique en choisissant "**Automatically setup new a SSH key pair for Ionic Pro**"

Suivez ensuite le setup et garder les valeurs par défaut (choisir "**Y**" à chaque fois).

La syntaxe générique d'une création d'application est la suivante :

```
$ ionic start [<name>] [<template>]
```

Entrée	Description
name	C'est le nom de votre application au format Camel par ex. Vous pouvez également l'écrire tout en minuscule (ce que je recommande)
template	C'est le template ionic de votre choix. Pour afficher la liste des templates disponible actuellement, vous pouvez saisir la commande ionic start --list (voir ci-dessous)

```
$ ionic start --list
tabs ..... ionic-angular A starting project with a simple tabbed interface
blank ..... ionic-angular A blank starter project
```

```

sidemenu ..... ionic-angular A starting project with a side menu with navigation in the content area
super ..... ionic-angular A starting project complete with pre-built pages, providers and best practices for Ionic development.
conference ..... ionic-angular A project that demonstrates a realworld application
tutorial ..... ionic-angular A tutorial based project that goes along with the Ionic documentation
aws ..... ionic-angular AWS Mobile Hub Starter
tabs ..... ionic1 A starting project for Ionic using a simple tabbed interface
blank ..... ionic1 A blank starter project for Ionic
sidemenu ..... ionic1 A starting project for Ionic using a side menu with navigation in the content area
maps ..... ionic1 An Ionic starter project using Google Maps and a side menu

```

il est également possible de créer une application à partir d'un dépôt git :

```
$ ionic start monappli_sur_git https://github.com/charlesen/monappli_sur_git
```

Une fois votre application créée, accédez au dossier nouvellement créé, puis démarrer le projet :

```

$ cd monappli
$ ionic serve -lc

```

Ionic devrait ensuite ouvrir votre application depuis votre navigateur préféré.

Création du projet Duckcoin

Comme nous l'avons vu, il est possible de créer une application mobile à partir d'un dépôt git. C'est ce que nous allons faire pour l'application DuckCoin.

Ouvrez donc votre terminal et saisissez les commandes suivantes :

```
$ ionic start duckcoin https://github.com/charlesen/duckcoin-starter
```

```

:~/www/books/ionic$ ionic start duckcoin https://github.com/charlesen/duckcoin
> git clone https://github.com/charlesen/duckcoin duckcoin --progress
Clonage dans 'duckcoin'...
remote: Counting objects: 114, done.
remote: Compressing objects: 100% (99/99), done.
remote: Total 114 (delta 8), reused 110 (delta 8), pack-reused 0
Récception d'objets: 100% (114/114), 1.37 MiB | 852.00 KiB/s, fait.
Résolution des deltas: 100% (8/8), fait.
Vérification de la connectivité...
fait.

Installing dependencies may take several minutes.

*  IONIC DEVAPP *
Speed up development with the Ionic DevApp, our fast, on-device testing mobile app
- 🏁 Test on iOS and Android without Native SDKs
- 🎨 LiveReload for instant style and JS updates
-->   Install DevApp: https://bit.ly/ionic-dev-app   <--
> npm i

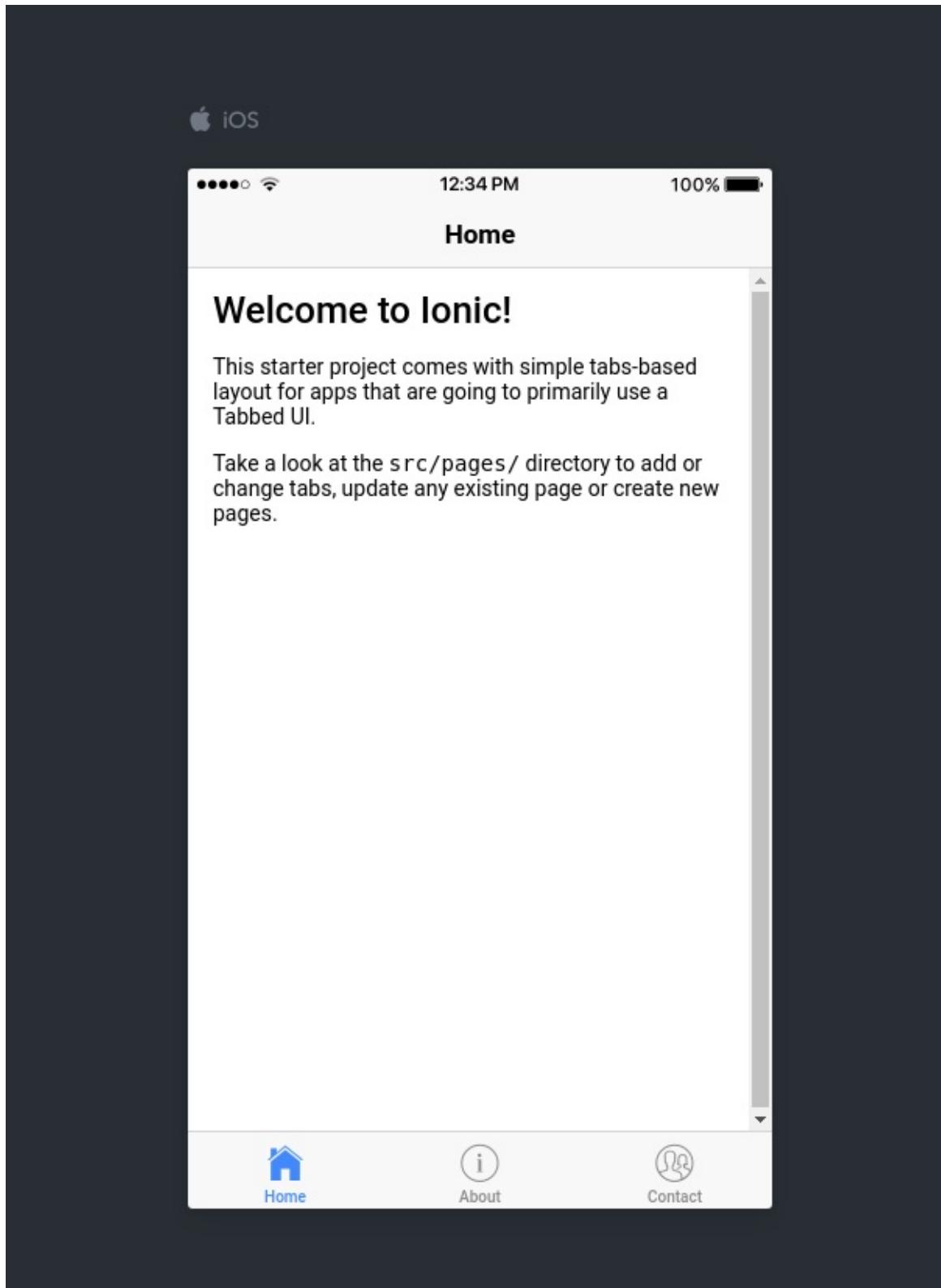
```

Démarrez ensuite l'application avec la commande serve :

```

$ cd duckcoin
$ ionic serve -lc

```



Structure du projet

Faisons un peu le tour de l'anatomie d'un projet type sous Ionic.

./src/index.html

C'est l'entrée principale du projet. Il faut se rappeler qu'une application Hybride utilise la technologie WebView du téléphone qui se comporte comme un mini-navigateur à l'intérieur duquel on peut afficher un site web, qui est votre projet.

A l'intérieur de ce fichier, Ionic va aller chercher le tag **<ion-app>** à l'intérieur duquel vos différents écrans seront chargés.

```
<ion-app></ion-app>
```

On retrouve également du code javascript, généré par Ionic et qu'il ne sera pas nécessaire de modifier :

```
<!-- Ionic's root component and where the app will load -->
<ion-app></ion-app>

<!-- The polyfills js is generated during the build process -->
<script src="build/polyfills.js"></script>

<!-- The vendor js is generated during the build process
     It contains all of the dependencies in node_modules -->
<script src="build/vendor.js"></script>

<!-- The main bundle js is generated during the build process -->
<script src="build/main.js"></script>
```

./src/

C'est à l'intérieur que l'on retrouve le code de l'application à proprement. Lorsque l'on voudra rajouter de nouveaux écrans, de la logique métier,... c'est ici que cela se passera.

On retrouve du code écrit en TypeScript (nous en reparlerons en détails au [chapitre 6](#)) dont l'extension de fichiers est **.ts**.

On retrouve aussi du html, du css,...

Ecran Mobile = 1 Fichier .ts + 1 Fichier .html + 1 Fichier .scss

le fichier **src/app/app.module.ts** est le point d'entrée métier de notre application.

```
@NgModule({
  declarations: [MyApp, ContactPage, HomePage],
  imports: [BrowserModule, IonicModule.forRoot(MyApp)],
  bootstrap: [IonicApp],
  entryComponents: [MyApp, ContactPage, HomePage],
  providers: []
})
export class AppModule {}
```

C'est dans ce fichier que l'on décide quelle composant (ici `MyApp`) sera le composant principal. On expliquera ces notions de composants dans le [chapitre 8](#).

Dans le [chapitre suivant](#), nous allons apprendre à customiser notre application pour qu'elle soit un peu plus à notre image. Mais en attendant, exercez-vous un peu.

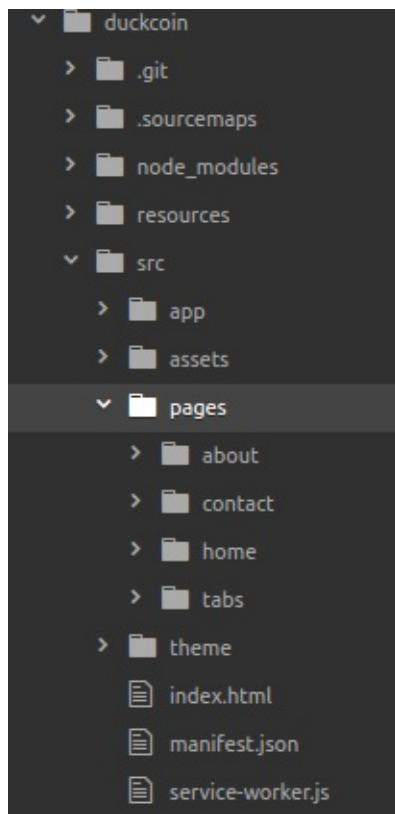
Exercez-vous

1) Créez, si ce n'est déjà fait, l'application duckcoin comme expliqué précédemment

```
$ ionic start duckcoin https://github.com/charlesen/duckcoin-starter
```

```
$ cd duckcoin
```

2) Faites le tour de l'application pour découvrir un petit peu son architecture



3) Renommer les différents onglets :

- Home en **Accueil**
- About en **Minage**
 - Contact en **Portefeuille**
 - Que se passe t-il dans la console à chaque enregistrement ?

4) Faites les modifications suivantes dans les onglets

- **Accueil** : changez le contenu de l'onglet par le contenu de la page d'accueil du site <https://duckcoin.charlesen.fr>. Et renommer la page, de **Home** à **Duckcoin**. (**src/pages/home/home.html**). Les images sont à placer dans le dossier **src/assets/imgs**.
- **Minage** : Changer l'intitulé de la page en **Minage**.
- **Portefeuille** : Changer l'intitulé de la page en **Portefeuille**.

5) Editez le fichier **theme/variables.scss** et modifier le contenu de la façon suivante :

```

// Named Color Variables
// -----
// Named colors makes it easy to reuse colors on various components.
// It's highly recommended to change the default colors
// to match your app's branding. Ionic uses a Sass map of
// colors so you can add, rename and remove colors as needed.
// The "primary" color is the only required color in the map.

$colors: (
  primary:    #488aff,

```

```

secondary: #32db64,
danger: #f53d3d,
light: #f4f4f4,
dark: #222,
duckcoin : #df4932 // <!-- ICI
);

```

Enregistrez, puis dans le fichier **src/pages/home/home.html**, effectuez les modifications suivantes

```

<ion-header>
  <ion-navbar color="duckcoin"><!-- ICI -->
    <ion-title>Home</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  LE CONTENU QUE VOUS AVEZ MODIFIÉ JUSTE AVANT ;-)
</ion-content>

```

Que s'est-il passé ?

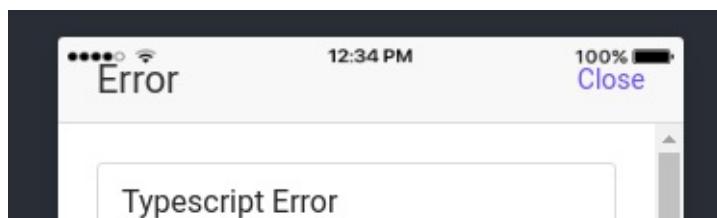
Faites la même chose pour l'ensemble des onglets.

6) Adaptez le code hexadécimal de la couleur duckcoin pour qu'il soit le plus proche de vos goûts. Le meilleur code couleur sera utilisé dans la suite du projet :-)

7) Renommez les fichiers **about.html** en **mining.html**, **about.scss** en **mining.scss**, **about.scss** en **mining.scss**, **about.ts** en **mining.ts**. Puis, renommer le dossier **about** (src/pages/about) en **mining** (src/pages/mining).

Dans le fichier **mining.ts**, remplacez **AboutPage** par **MiningPage**.

Que se passe-t-il dans la console ? Dans votre navigateur ? Quelles solutions proposeriez-vous ? Voir par exemple le contenu du fichier src/pages/mining/mining.ts.



8) Editer le fichier **src/app/app.module.ts** de manière à corriger le maximum d'erreurs.

9) Effectuez les actions précédentes pour l'onglet Portefeuille (renommage + resolutions de bugs) : **contact.html** en **wallet.html**, **contact.scss** en **wallet.scss**, **contact.scss** en **wallet.scss**, **contact.ts** en **wallet.ts**.



10) Ajustez le style CSS de la page d'accueil pour rendre le contenu de l'onglet un peu plus joli : **src/pages/home/home.scss**. N'hésitez pas utiliser l'inspecteur de votre navigateur (F12).

11) Nous allons à présent lier notre application au service Ionic PRO. Dans votre invite de commandes, faites :

```
$ ionic link
✓ Looking up your apps - done!

? Which app would you like to link (Use arrow keys)
❯ Create a new app
  Duckcoin-starter (94d675be)
  Duckcoin (20e8461e)
```

Dans la liste qui vous est proposée, choisir l'application que vous avez créée depuis votre espace Ionic PRO.

Il vous sera également proposé d'héberger votre application soit sur Github, soit sur le cloud Ionic. Pour les raisons de ce TP nous utiliserons Ionic PRO, mais vous pouvez très bien aussi utiliser github (votre code source sera alors public).

```
> ionic git remote
> git remote add ionic git@git.ionicjs.com:charlesen/duckcoin-starter.git
[OK] Added remote ionic.
[OK] Project linked with app 94d675be!
```

Editez ensuite le fichier **ionic.config.json**. Que remarquez-vous ?

1. Ubuntu Ionic Installer : https://github.com/nraboy/ubuntu-ionic-installer/blob/master/ubuntu_ionic_installer.sh ↵
2. *How to prevent permission errors* : <https://docs.npmjs.com/getting-started/fixing-npm-permissions> ↵

Chap 4 - Templates et Customisation

Dans le chapitre précédent nous avons installé Ionic et ses dépendances. Nous avons également pu créer notre application Duckcoin et nous nous sommes amusés à le modifier tant bien que mal.



Dans ce chapitre, nous allons apprendre à customiser un peu plus notre application mobile et à créer de nouvelles pages.

Customisation

Attributs de style

Ionic met à disposition un ensemble d'attributs utilitaires qui peuvent être utilisés sur n'importe quel élément afin de modifier du texte, le centrer par exemple, ou encore gérer les marges. A la différence de Bootstrap où on fait usage de classes css (row, col,...), ici on utilisera plutôt des attributs.

	Style CSS	
--	-----------	--

Attribut	correspondant	Description
text-left	text-align: left	Aligne du texte à gauche
text-right	text-align:right	Aligne du texte à droite
text-start	text-align:start	Identique à text-left si la direction d'écriture va de gauche vers la droite et text-right si la direction est de droite vers la gauche.
text-end	text-align:end	Identique à text-right si la direction d'écriture est de gauche vers la droite et de text-left si la direction est de droite vers la gauche.
text-center	text-align:center	Centre le contenu
text-justify	text-align:justify	Justifie le contenu

Une liste beaucoup plus exhaustive se trouve dans la documentation qui est extrêmement bien faite

<https://ionicframework.com/docs/theming/css-utilities/>

On va pouvoir utiliser ces attributs directement dans nos pages. Centrons par exemple le h2 de la page d'accueil et justifions le contenu du texte qui le suit :

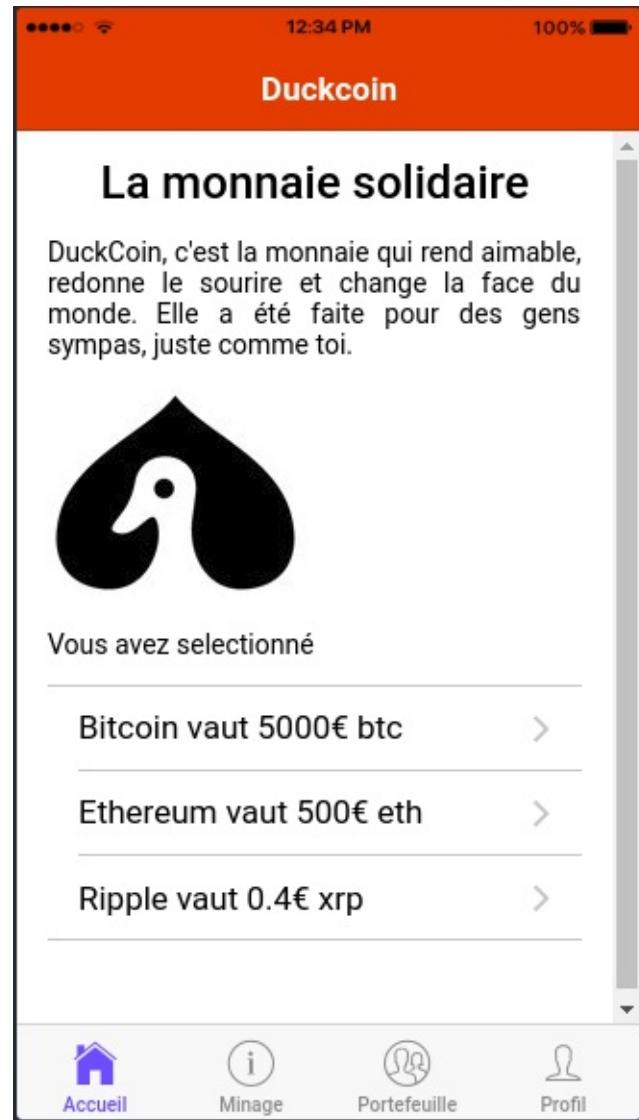
src/pages/home/home.html

```

<ion-header>
  <ion-navbar color="duckcoin">
    <ion-title>Duckcoin</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <h2 text-center>La monnaie solidaire</h2> <!-- ICI -->
  <p text-justify> <!-- Et Là -->
    DuckCoin, c'est la monnaie qui rend aimable, redonne le sourire et change la face du monde.
    Elle a été faite pour des gens sympas, juste comme toi.
  </p>
  ...
</ion-content>

```



Grille CSS responsive

Ionic propose également un système de grille css pour permettre une meilleur gestion de blocs de contenus. Il est assez similiare dans sa syntaxe à celui que propose Bootstrap.

```

<h2>Profil utilisateur</h2>
<ion-grid>
  <ion-row>
    <ion-col col-lg-1>
      
    </ion-col>
    <ion-col>
      Charles E.
    </ion-col>
    <ion-col>
      Développeur Web et Mobile
    </ion-col>
  </ion-row>
  <ion-row>
    <ion-col>
      Intérêt pour la finance, la blockchain, les technologies mobiles et le 0'tacos
    </ion-col>
  </ion-row>
</ion-grid>

```



Pour plus de détails, merci de consulter la documentation correspondante :

<https://ionicframework.com/docs/theming/responsive-grid/>

Utilisation de SASS

Ionic est construit sur Sass (Syntactically Awesome Stylesheets), un langage de génération de feuilles de style, robuste et facile à prendre en main. En fait si vous savez déjà définir une feuille de style, ce langage ne vous choquera pas trop. Grâce à cette technologie embarquée dans Ionic, nous allons non seulement pouvoir définir des styles génériques pour notre application, qui pourront être utilisé à plusieurs endroits différents, mais nous pourrons également changer les styles par défaut des attributs et composants Ionic.

La définition ou la redéfinition de style css dynamique se fait depuis le fichier **src/theme/variables.scss** :

```
// Named Color Variables
// -----
// Named colors makes it easy to reuse colors on various components.
// It's highly recommended to change the default colors
// to match your app's branding. Ionic uses a Sass map of
// colors so you can add, rename and remove colors as needed.
// The "primary" color is the only required color in the map.
```

```
$colors: (
  primary:    #488aff,
  secondary:  #32db64,
  danger:    #f53d3d,
  light:     #f4f4f4,
  dark:      #222,
  duckcoin:   #df4932 // Notre première valeur SASS
);
```

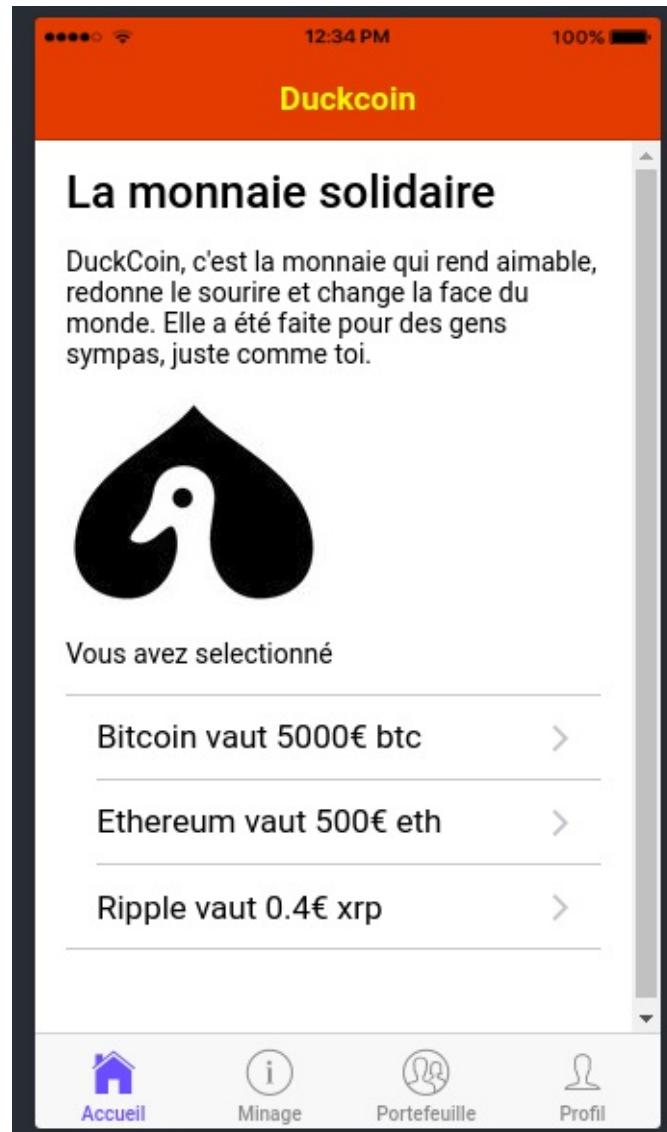
Vous pouvez ici effectuer des changements sur les valeurs par défaut des thèmes primaire, secondaire, ...Et ils s'appliqueront automatiquement à l'ensemble de vos composants.

```
<ion-header>
  <ion-navbar color="duckcoin">
    <ion-title>Duckcoin</ion-title>
  </ion-navbar>
</ion-header>
```

Ici la barre de navigation aura comme couleur de fond (background) celle définie dans le fichier de variables scss et comme couleur de texte du blanc. Si vous voulez autre chose que du blanc, disons du jaune, vous devriez modifier votre style comme ceci :

src/theme/variables.scss

```
$colors: (
  primary:    #488aff,
  secondary:  #32db64,
  danger:    #f53d3d,
  light:     #f4f4f4,
  dark:      #222,
  duckcoin: (
    base: #df4932,
    contrast: yellow
  )
);
```



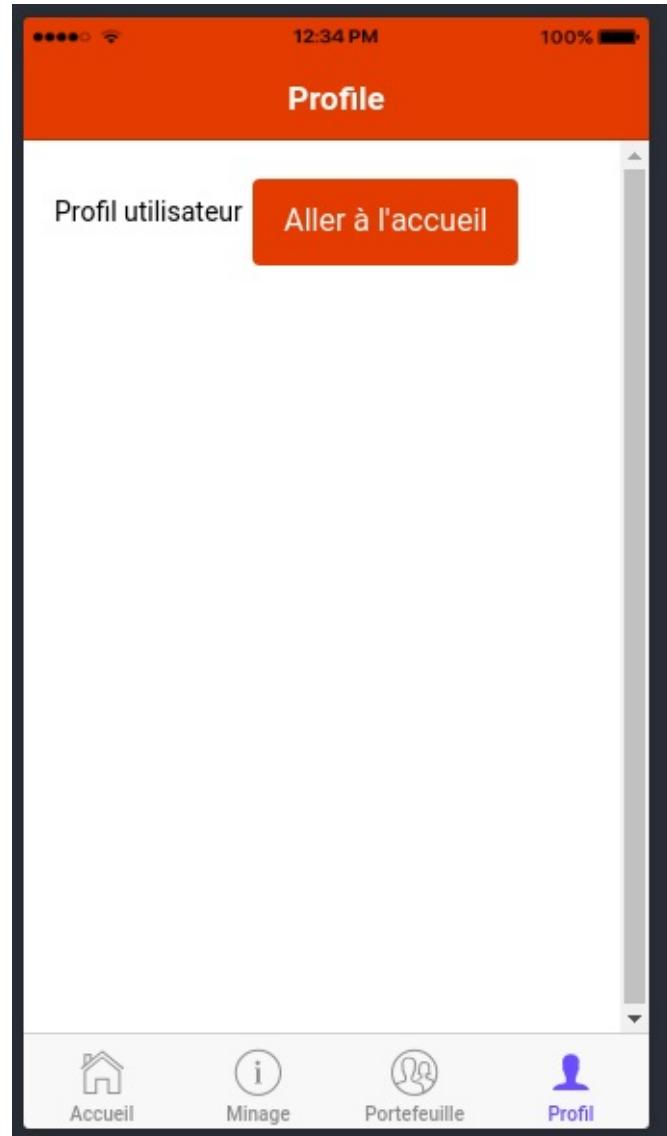
Il est également possible d'appeler des variables définis dans ce fichier **src/theme/variables.scss** directement dans nos fichiers scss. Changeons par exemple la couleur du bouton présent dans l'onglet Profil :

src/pages/profile/profile.scss

```
page-profile {  
  button[ion-button] {  
    background: color($colors, duckcoin, base);  
  }  
}
```

qui est l'équivalent css de :

```
page-profile {  
  button[ion-button] {  
    background: #df4932;  
  }  
}
```



De manière générale, les Variables Sass vous permettent de définir une valeur une fois, puis de l'utiliser à plusieurs endroits différents. Une variable commence toujours par un signe dollar (\$) et est initialisé comme une propriété CSS classique.

Supposons par exemple que l'on souhaiterait imposer une largeur maximale sur un certain nombre de composants de notre application (des images, boutons,...). On pourrait par exemple faire ceci dans le fichier variables.css :

src/theme/variables.scss

```
$max-width: 400px;
```

Puis dans une ou plusieurs feuilles de style scss invoquer notre variable :

```
div {
  width:$max-width;
}
```

On pourrait même faire des calculs sur la variable :

```
img {
  width : $max-width/10;
```

```
}
```

Templates et création de nouvelles pages

Racine de toutes les pages

Considérons le fichier **src/app/app.html**, c'est à partir de ce fichier que sera rendu toutes les autres pages.

```
<ion-nav [root]="'rootPage"></ion-nav>
```

On y définit un paramètre **rootPage** qui sera en fait le composant à afficher par défaut, une sorte d'index. Ce paramètre **rootPage** est lui-même déclaré dans le fichier **app.component.ts**.

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

import { TabsPage } from '../pages/tabs/tabs';

@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage: any = TabsPage; // <!-- ICI

  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }
}
```

On pourrait tout à fait remplacer le **rootPage** par une autre page, la page de Minage par exemple.

```
import { Component } from '@angular/core';
import { Platform } from 'ionic-angular';
import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';

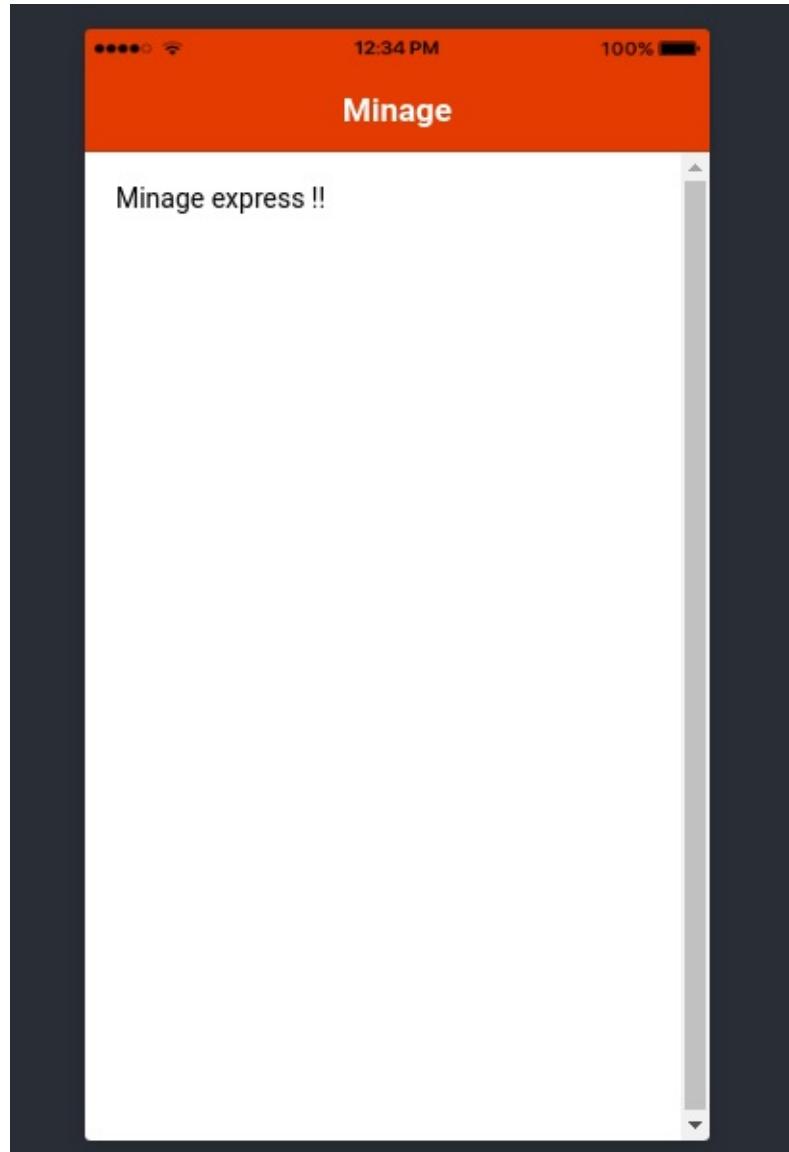
import { TabsPage } from '../pages/tabs/tabs';
import { MiningPage } from '../pages/mining/mining'; // <!-- ICI

@Component({
  templateUrl: 'app.html'
})
export class MyApp {
  rootPage: any = MiningPage; // <!-- et là

  constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
    platform.ready().then(() => {
      // Okay, so the platform is ready and our plugins are available.
      // Here you can do any higher level native things you might need.
      statusBar.styleDefault();
      splashScreen.hide();
    });
  }
}
```

Ce qui ferait que par défaut, lorsque votre application se lancera on aura par défaut cette page de Minage au lieu de la page affichant des onglets.



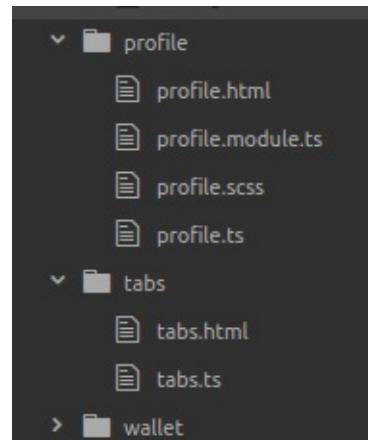


Création d'une nouvelle page

Pour créer une nouvelle page, il vous suffit de saisir la commande **ionic g page LeNomDeLaPage** :

```
$ ionic g page Profile
[OK] Generated a page named Profile!
```

Dans cet exemple, j'ai créé une nouvelle page pour afficher un profil utilisateur. Cette commande m'a automatiquement générer le triplet : **fichier .ts + fichier .html + fichier .scss**.



profile.ts

```

import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular';

/**
 * Generated class for the ProfilePage page.
 *
 * See https://ionicframework.com/docs/components/#navigation for more info on
 * Ionic pages and navigation.
 */
@IonicPage()
@Component({
  selector: 'page-profile',
  templateUrl: 'profile.html',
})
export class ProfilePage {

  constructor(public navCtrl: NavController, public navParams: NavParams) {
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad ProfilePage');
  }
}

```

profile.html

```

<!--
Generated template for the ProfilePage page.

See http://ionicframework.com/docs/components/#navigation for more info on
Ionic pages and navigation.
-->
<ion-header>

  <ion-navbar>
    <ion-title>Profile</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>

</ion-content>

```

profile.scss

```
page-profile {  
}  
}
```

Ajoutons à présent cette nouvelle page à notre système d'onglet comme ceci :

src/pages/tabs/tabs.ts

```
import { Component } from '@angular/core';  
  
import { MiningPage } from '../mining/mining';  
import { WalletPage } from '../wallet/wallet';  
import { HomePage } from '../home/home';  
import { ProfilePage } from '../profile/profile'; // On importe la nouvelle page ICI  
  
@Component({  
  templateUrl: 'tabs.html'  
})  
export class TabsPage {  
  
  tab1Root = HomePage;  
  tab2Root = MiningPage;  
  tab3Root = WalletPage;  
  tab4Root = ProfilePage; // On crée le nouvel onglet  
  
  constructor() {  
  
  }  
}
```

src/pages/tabs/tabs.html

```
<ion-tabs>  
  <ion-tab [root]="tab1Root" tabTitle="Accueil" tabIcon="home"></ion-tab>  
  <ion-tab [root]="tab2Root" tabTitle="Minage" tabIcon="information-circle"></ion-tab>  
  <ion-tab [root]="tab3Root" tabTitle="Portefeuille" tabIcon="contacts"></ion-tab>  
  <!-- Affichage du nouvel onglet -->  
  <ion-tab [root]="tab4Root" tabTitle="Profil" tabIcon="person"></ion-tab>  
</ion-tabs>
```

Il faut ensuite déclarer cette nouvelle page dans le module principale, pour que la communauté des pages puissent le connaître et pouvoir éventuellement l'appeler si besoin. Pour cela, il vous faut modifier le fichier src/app/app.module.ts de la manière suivante :

```
import { NgModule, ErrorHandler } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';  
import { MyApp } from './app.component';  
  
import { MiningPage } from '../pages/mining/mining';  
import { WalletPage } from '../pages/wallet/wallet';  
import { HomePage } from '../pages/home/home';  
import { ProfilePage } from '../pages/profile/profile'; // On importe la nouvelle page ICI  
import { TabsPage } from '../pages/tabs/tabs';  
  
import { StatusBar } from '@ionic-native/status-bar';  
import { SplashScreen } from '@ionic-native/splash-screen';  
  
@NgModule({  
  declarations: [  
    MyApp,
```

```
  MiningPage,
  WalletPage,
  HomePage,
  ProfilePage, // On la déclare ici
  TabsPage
],
imports: [
  BrowserModule,
  IonicModule.forRoot(MyApp)
],
bootstrap: [IonicApp],
entryComponents: [
  MyApp,
  MiningPage,
  WalletPage,
  HomePage,
  ProfilePage, // Et là
  TabsPage
],
providers: [
  StatusBar,
  SplashScreen,
  {provide: ErrorHandler, useClass: IonicErrorHandler}
]
})
export class AppModule {}
```

Après enregistrement vous devriez voir ceci s'afficher à présent :



Navigation entre différentes pages

Pour passer d'une page, par exemple de la page de Profil, à la page d'accueil par exemple, on utilise ce que l'on appelle le contrôleur de navigation (NavController), que vous avez du voir apparaître dans chaque page.

src/pages/profile/profile.ts

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular'; // ICI

@IonicPage()
@Component({
  selector: 'page-profile',
  templateUrl: 'profile.html',
})
export class ProfilePage {

  constructor(public navCtrl: NavController, public navParams: NavParams) { // Et là
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad ProfilePage');
  }
}
```

```

    }
}
```

Dans ce fichier, ajoutez la fonction gotoHome suivante :

```

import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular'; // ICI

import { HomePage } from '../home/home'; // On importe la Page d'accueil

@IonicPage()
@Component({
  selector: 'page-profile',
  templateUrl: 'profile.html',
})
export class ProfilePage {

  constructor(public navCtrl: NavController, public navParams: NavParams) { // Et là
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad ProfilePage');
  }

  /**
   * Cette fonction permet d'aller à la page d'accueil
   */
  gotoHome() {
    this.navCtrl.push(HomePage, {
      un_parametre: 'Je suis un paramètre'
    });
  }
}
```

Puis, modifions un peu le fichier **src/pages/profile/profile.ts** pour afficher un bouton qui nous permettra d'appeler cette action :

```

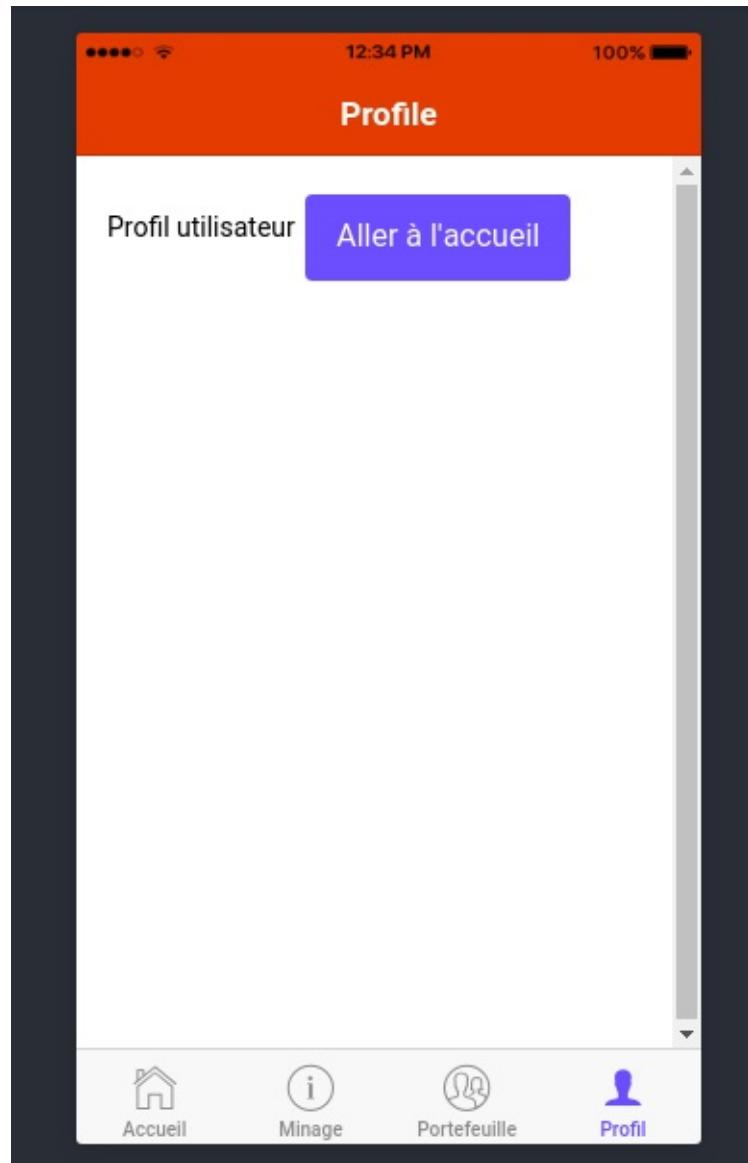
<ion-header>

  <ion-navbar color="duckcoin">
    <ion-title>Profile</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  Profil utilisateur
  <button ion-button (click)="gotoHome()">Aller à l'accueil</button>
</ion-content>
```

Qui donne le résultat suivant :





L'élément button fait partie des composants que propose Ionic. Dans le [chapitre suivant](#), nous ferons le tour de ces principaux composants et apprendrons à la customiser.

Exercez-vous

- 1) Créez la page Profile précédente et configuez là pour quelle soit dans le thème de l'application. Corrigez les bugs eventuels.
- 2) Dan la fonction **gotoHome**, remplacez "push" par "pop" : this.navCtrl.pop. Que constatez-vous ?
- 3) Editez le fichier **app.module.ts** de la manière suivante :

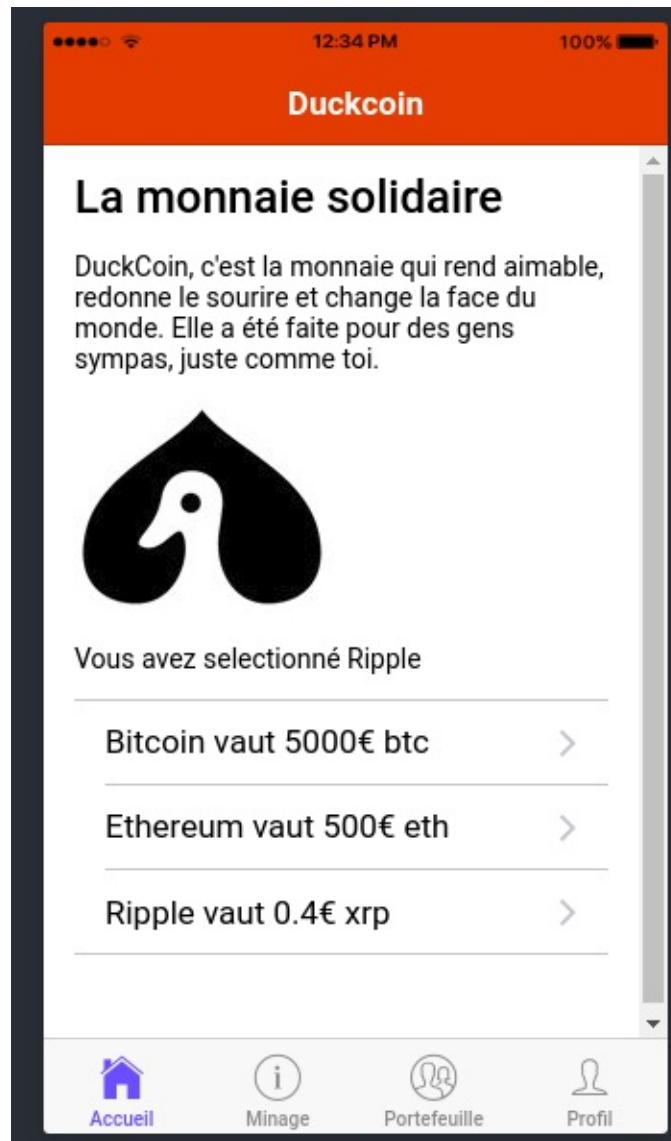
```
...
@NgModule({
  declarations: [
    MyApp,
    MiningPage,
    WalletPage,
    HomePage,
    ProfilePage, // On la déclare ici
    TabsPage
  ],
  ...
})
```

```
imports: [
  BrowserModule,
  IonicModule.forRoot(MyApp, {
    tabsPlacement: 'top',
    backButtonText: 'Retour'
  })
],
...
```

Que remarquez-vous ?

4) Allez à l'adresse suivante : <https://ionicframework.com/docs/components>

Comment à partir des informations qu'y s'y trouve peut-on rajouter une liste d'éléments en page d'accueil (voir screen suivant)



Astuces :

```
export class HomePage {
  selected : any = '';
  items : any = [];
  constructor(public navCtrl: NavController) {
    this.items = [
      {'title':'Bitcoin', 'currency':'btc', 'price':'5000€'},
      {'title':'Ethereum', 'currency':'eth', 'price':'500€'},
      {'title':'Ripple', 'currency':'xrp', 'price':'0.4€'}
    ];
  }
}
```

```
        {'title':'Ethereum', 'currency':'eth', 'price':'500€'},
        {'title':'Ripple', 'currency':'xrp', 'price':'0.4€'}
    ];
}

itemSelected(item) {
    this.selected =item;
}

}
```

5) Testez d'autres composants

6) Créez une page Setting et ajoutez à cette page un formulaire avec des éléments simples : nom, prénom, adresse,...

Chap 5 - Utilisation des composants Ionic

Ionic est constitué de blocs d'éléments de haut niveau appelés composants. Les composants vous permettent de créer rapidement une interface pour votre application.

Le framework propose pléthore de composants, du bouton au toast, en passant par des listes d'éléments, soit suffisamment d'UI pour développer à peu près tout type d'application.

Chaque composant a son propre API. Ce qui permet de l'exploiter au maximum. Etudions quelques-uns d'entre eux qui nous seront bien utiles.

Liste des composants : <https://ionicframework.com/docs/components/>

Faisons le tour de quelques composants intéressants.

Composant Bouton

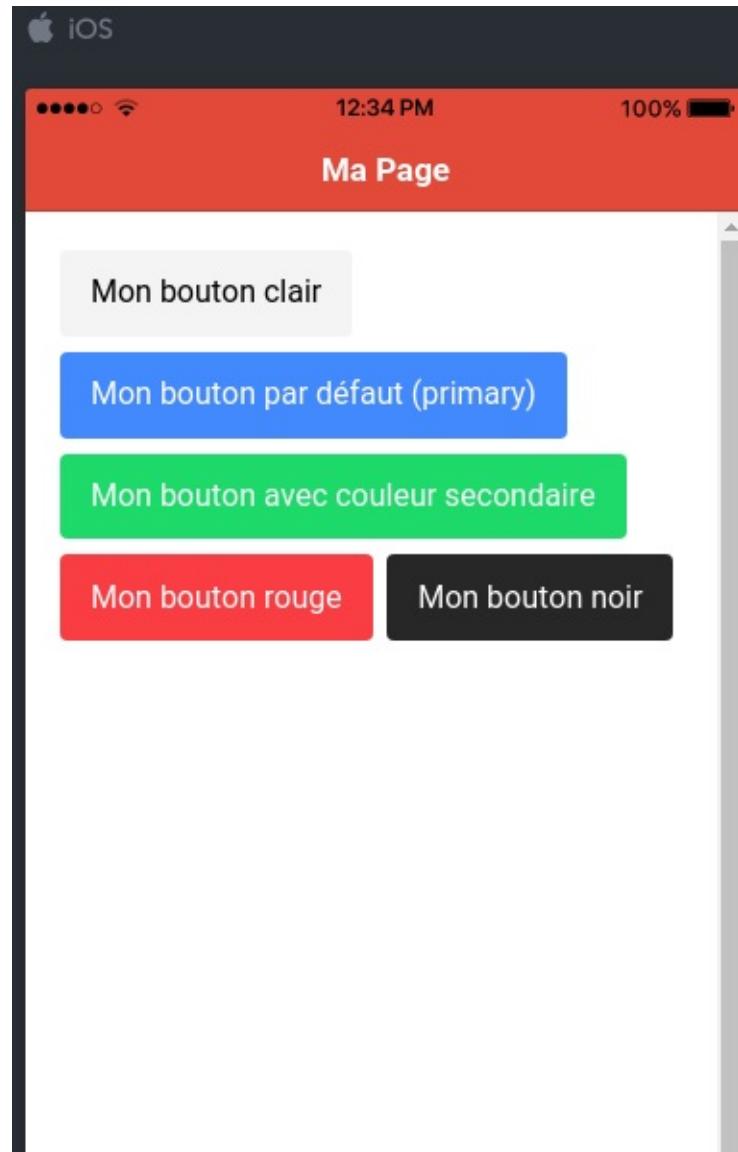
Pour ajouter un composant de type bouton à votre application mobile, il suffit simplement de faire (attention, grand moment) :

```
<button ion-button>Mon bouton</button>
```

Tada ! Ce n'est pas plus compliqué que ça.

On peut aussi customiser un peu ce bouton grâce à des [directives](#), concept abordé au [chapitre 8](#). Ajustons par exemple la couleur des différents boutons :

```
<button ion-button color="light">Mon bouton clair</button>
<button ion-button>Mon bouton par défaut (primary)</button>
<button ion-button color="secondary">Mon bouton avec couleur secondaire</button>
<button ion-button color="danger">Mon bouton rouge</button>
<button ion-button color="dark">Mon bouton noir</button>
```



on peut également retirer le background pour n'afficher que la bordure dans la couleur définie grâce à la directive **outline** :

```
<ion-header>

  <ion-navbar color="duckcoin">
    <ion-title>Ma Page</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <button ion-button outline>Mon bouton par défaut (primary)</button>
  <button ion-button color="secondary" outline>Mon bouton avec couleur secondaire</button>
  <button ion-button color="danger" outline>Mon bouton rouge</button>
  <button ion-button color="dark" outline>Mon bouton noir</button>
</ion-content>
```



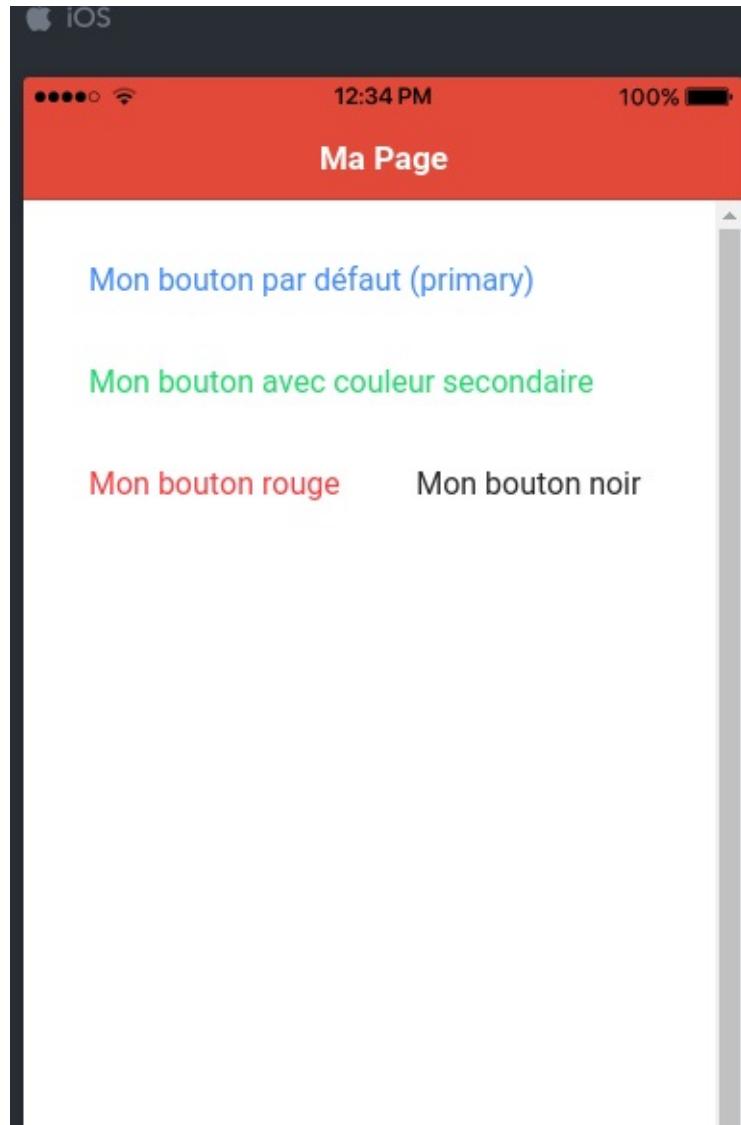
Ou encore retirer les bordures du boutons avec la directive **clear** :

```
<ion-header>

  <ion-navbar color="duckcoin">
    <ion-title>Ma Page</ion-title>
  </ion-navbar>

</ion-header>

<ion-content padding>
  <button ion-button clear>Mon bouton par défaut (primary)</button>
  <button ion-button color="secondary" clear>Mon bouton avec couleur secondaire</button>
  <button ion-button color="danger" clear>Mon bouton rouge</button>
  <button ion-button color="dark" clear>Mon bouton noir</button>
</ion-content>
```



Documentation : <https://ionicframework.com/docs/components/#buttons>

Composant Liste

Comme son nom peut le suggérer, ce composant va nous permettre d'ajouter une liste d'éléments à notre application. Affichons par exemple la liste des principales cryptomonnaies en avril 2018 :

src/pages/mapage/mapage.html

```
<ion-header>
  <ion-navbar color="duckcoin">
    <ion-title>Ma Page</ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <ion-list>
    <ion-list-header text-center>
      TOP 10 des cryptos en Avril 2018
    </ion-list-header>
    <ion-item *ngFor="let crypto of cryptos">
      <strong>{{ crypto.name }}</strong> vaut environ {{ crypto.price}}
    </ion-item>
  </ion-list>
```

```
</ion-content>
```

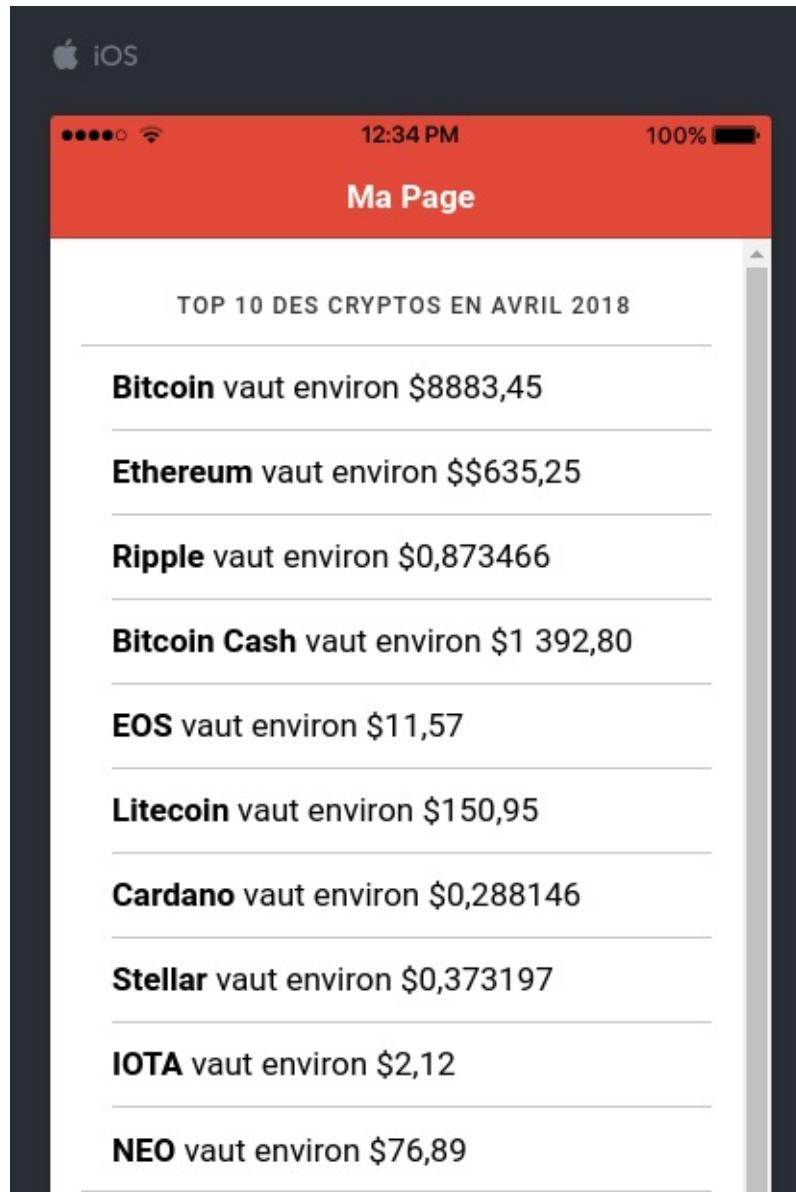
src/pages/mapage/mapage.ts

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular';

@IonicPage()
@Component({
  selector: 'page-mapage',
  templateUrl: 'mapage.html',
})
export class MaPagePage {
  cryptos: any = [];
  constructor(public navCtrl: NavController, public navParams: NavParams) {
    this.cryptos = [
      { 'name': 'Bitcoin', 'price': '$8883,45' },
      { 'name': 'Ethereum', 'price': '$$635,25' },
      { 'name': 'Ripple', 'price': '$0,873466' },
      { 'name': 'Bitcoin Cash', 'price': '$1 392,80' },
      { 'name': 'EOS', 'price': '$11,57' },
      { 'name': 'Litecoin', 'price': '$150,95' },
      { 'name': 'Cardano', 'price': '$0,288146' },
      { 'name': 'Stellar', 'price': '$0,373197' },
      { 'name': 'IOTA', 'price': '$2,12' },
      { 'name': 'NEO', 'price': '$76,89' }
    ];
  }

  ionViewDidLoad() {
    console.log('ionViewDidLoad SettingPage');
  }
}
```

Qui donne :



Documentation : <https://ionicframework.com/docs/components/#lists>

Composant Select

Le composant est similaire au tag html <select></select> et va donc nous permettre d'afficher une liste de choix. Affichons par exemple ici la liste des 5 premières cryptomonnaies par capitalisation boursière :

```

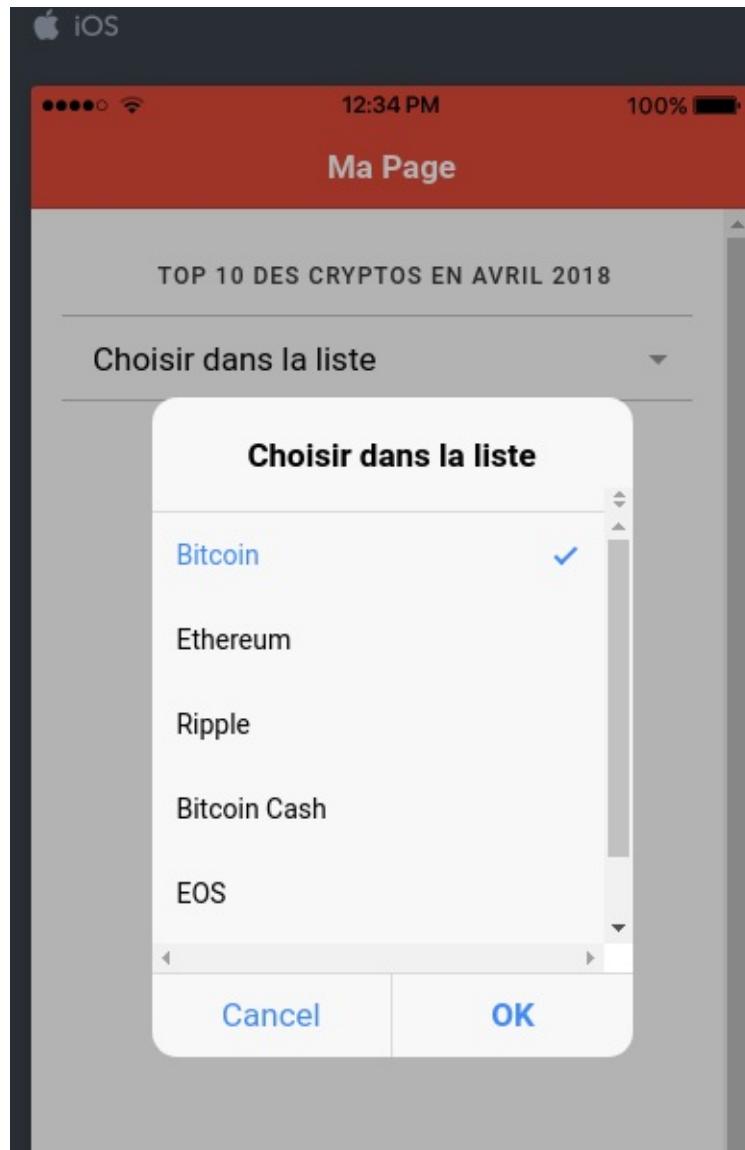
<ion-header>
  <ion-navbar color="duckcoin">
    <ion-title>Ma Page</ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <ion-list>
    <ion-list-header text-center>
      TOP 10 des cryptos en Avril 2018
    </ion-list-header>
    <ion-item>
      <ion-label>Choisir dans la liste</ion-label>
      <ion-select [(ngModel)]="gaming">
        <ion-option value="btc">Bitcoin</ion-option>

```

```

<ion-option value="xrp">Ethereum</ion-option>
<ion-option value="xrp">Ripple</ion-option>
<ion-option value="bch">Bitcoin Cash</ion-option>
<ion-option value="eos">EOS</ion-option>
<ion-option value="ltc">Litecoin</ion-option>
</ion-select>
</ion-item>
</ion-list>
</ion-content>

```



Documentation : <https://ionicframework.com/docs/components/#select>

Composant Cards

Les cartes sont un bon moyen d'afficher des informations importantes à destination des utilisateurs. Ce pattern s'inspire des carte de visite que nous utilisons dans la vie courante.

```

<ion-header>

<ion-navbar color="duckcoin">
  <ion-title>Ma Page</ion-title>
</ion-navbar>

```

```

</ion-header>

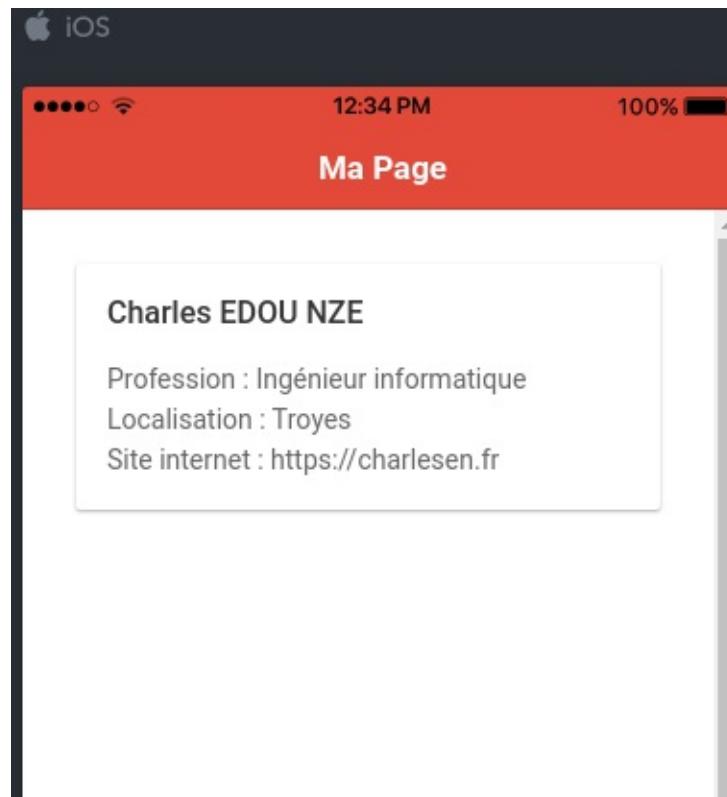
<ion-content padding>
  <ion-card>

    <ion-card-header>
      Charles EDOU NZE
    </ion-card-header>

    <ion-card-content>
      <p>Profession : Ingénieur informatique</p>
      <p>Localisation : Troyes</p>
      <p>Site internet : https://charlesen.fr</p>
    </ion-card-content>

  </ion-card>
</ion-content>

```



Il est également possible de combiner carte et liste d'éléments comme ceci :

```

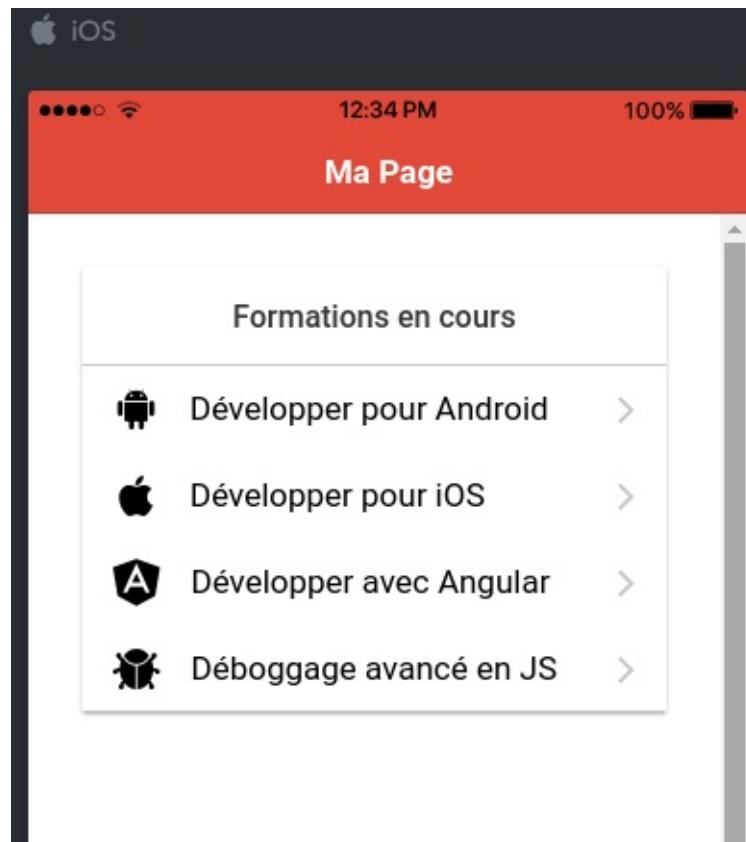
<ion-header>
  <ion-navbar color="duckcoin">
    <ion-title>Ma Page</ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <ion-card>
    <ion-card-header text-center>
      Formations en cours
    </ion-card-header>
    <ion-list>
      <button ion-item>
        <ion-icon name="logo-android" item-start></ion-icon>
        Développer pour Android
      </button>
    </ion-list>
  </ion-card>
</ion-content>

```

```

<button ion-item>
  <ion-icon name="logo-apple" item-start></ion-icon>
  Développer pour iOS
</button>
<button ion-item>
  <ion-icon name="logo-angular" item-start></ion-icon>
  Développer avec Angular
</button>
<button ion-item>
  <ion-icon name="bug" item-start></ion-icon>
  Déboggage avancé en JS
</button>
</ion-list>
</ion-card>
</ion-content>

```



Ou tout simplement reproduire un design assez proche de réseaux sociaux comme Instagram :

```

<ion-header>
  <ion-navbar color="duckcoin">
    <ion-title>Ma Page</ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <ion-card>
    
    <ion-card-content>
      <ion-card-title>
        DuckCoin
      </ion-card-title>
      <p>
        C'est la monnaie qui rend aimable, redonne le sourire et change la face du monde. Elle a été faite pour
        des gens sympas, juste comme toi.
      </p>
    </ion-card-content>
  </ion-card>

```

```
<ion-row no-padding>
  <ion-col>
    <button ion-button clear small color="duckcoin" icon-medium icon-start>
      J'aime <ion-icon name='heart'></ion-icon>
    </button>
  </ion-col>
  <ion-col text-right>
    <button ion-button clear small color="duckcoin" icon-medium icon-start>
      Partager <ion-icon name='share-alt'></ion-icon>
    </button>
  </ion-col>
</ion-row>
</ion-card>
</ion-content>
```



Documentation : <https://ionicframework.com/docs/components/#cards>

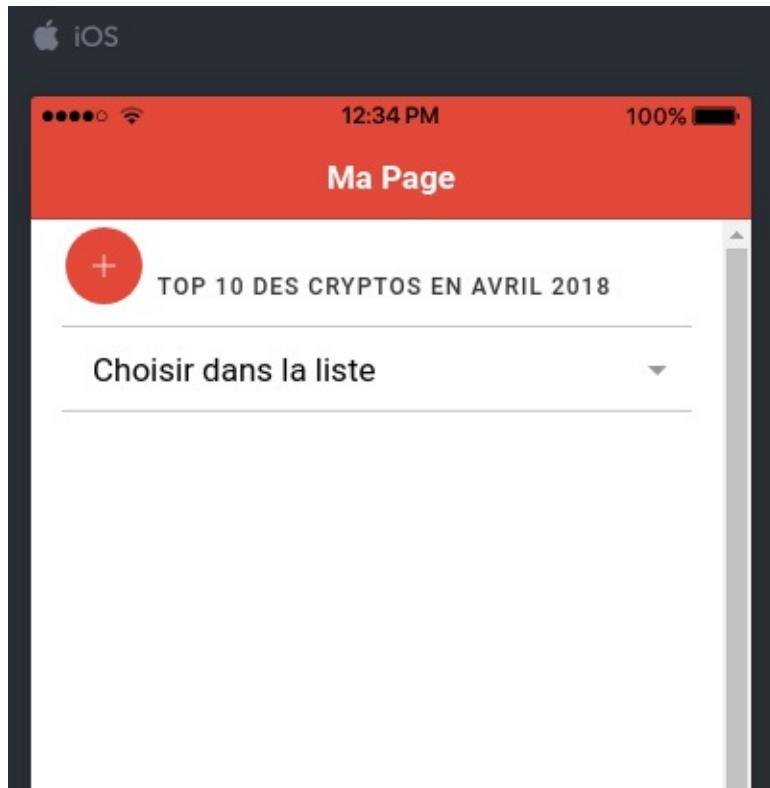
Composant Fab (Floating Action Buttons)

Ce composant est issu du Material Design, un ensemble de règles de design proposées par Google et qui est utilisé notamment à partir de la version 5.0 du système d'exploitation Android. Il s'agit d'un élément en forme de cercle qui permet, au clic, d'afficher d'autres éléments actionnables eux aussi par clic.

```
<ion-fab bottom right>
  <button ion-fab mini><ion-icon name="add"></ion-icon></button>
  <ion-fab-list side="top">
    <button ion-fab><ion-icon name="contacts"></ion-icon></button>
    <button ion-fab><ion-icon name="send"></ion-icon></button>
  </ion-fab-list>
</ion-fab>
```

On peut choisir d'afficher notre fab sur une tout au position que en bas à droite (**bottom right**). Affichons-le par exemple en haut à gauche

```
<ion-fab top left>
  <button ion-fab mini><ion-icon name="add"></ion-icon></button>
  <ion-fab-list side="top">
    <button ion-fab><ion-icon name="contacts"></ion-icon></button>
    <button ion-fab><ion-icon name="send"></ion-icon></button>
  </ion-fab-list>
</ion-fab>
```



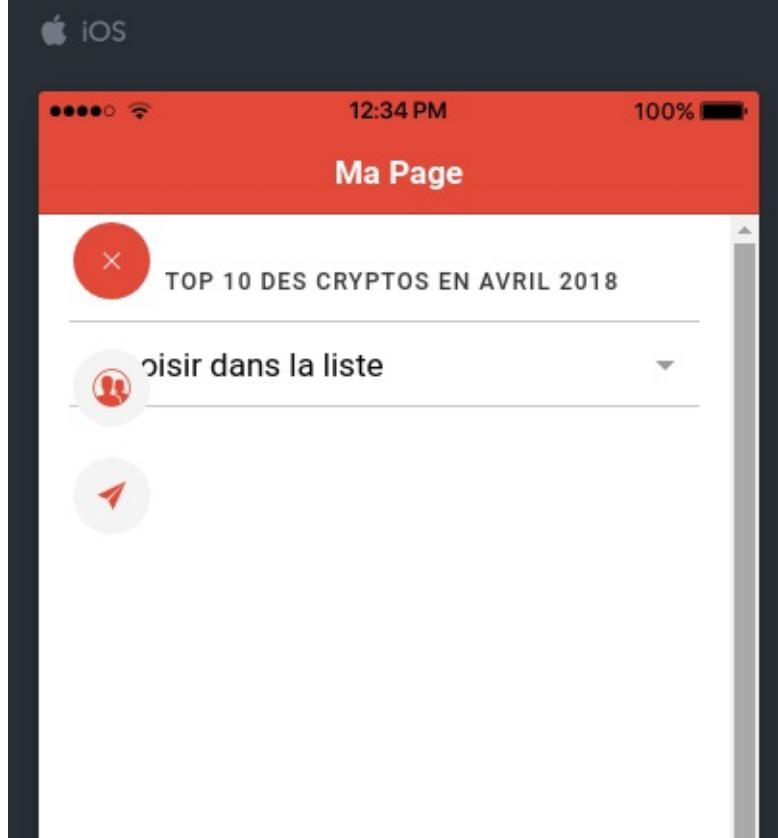
N'hésitez pas au besoin à adapter votre style scss comme ci :

src/pages/mapage.scss

```
ion-fab[top] {
  top:60px;
}
```

De plus, pour modifier la direction d'affichage de la liste d'action, il suffit de modifier la valeur du paramètre side à l'intérieur du tag **<ion-fab-list>**.

```
<ion-fab top left>
  <button ion-fab mini><ion-icon name="add"></ion-icon></button>
  <ion-fab-list side="bottom">
    <button ion-fab><ion-icon name="contacts"></ion-icon></button>
    <button ion-fab><ion-icon name="send"></ion-icon></button>
  </ion-fab-list>
</ion-fab>
```



Documentation : <https://ionicframework.com/docs/api/components/fab/FabButton/>

Exercez-vous

Dans cette série d'exercices, nous allons pouvoir améliorer un peu notre application. On va s'inspirer du design d'une application mobile nommée Electroneum.

Electroneum est une cryptomonnaie dédiée au secteur du mobile. Une application de la cryptomonnaie existe et est disponible pour Android et prochainement pour iOS.

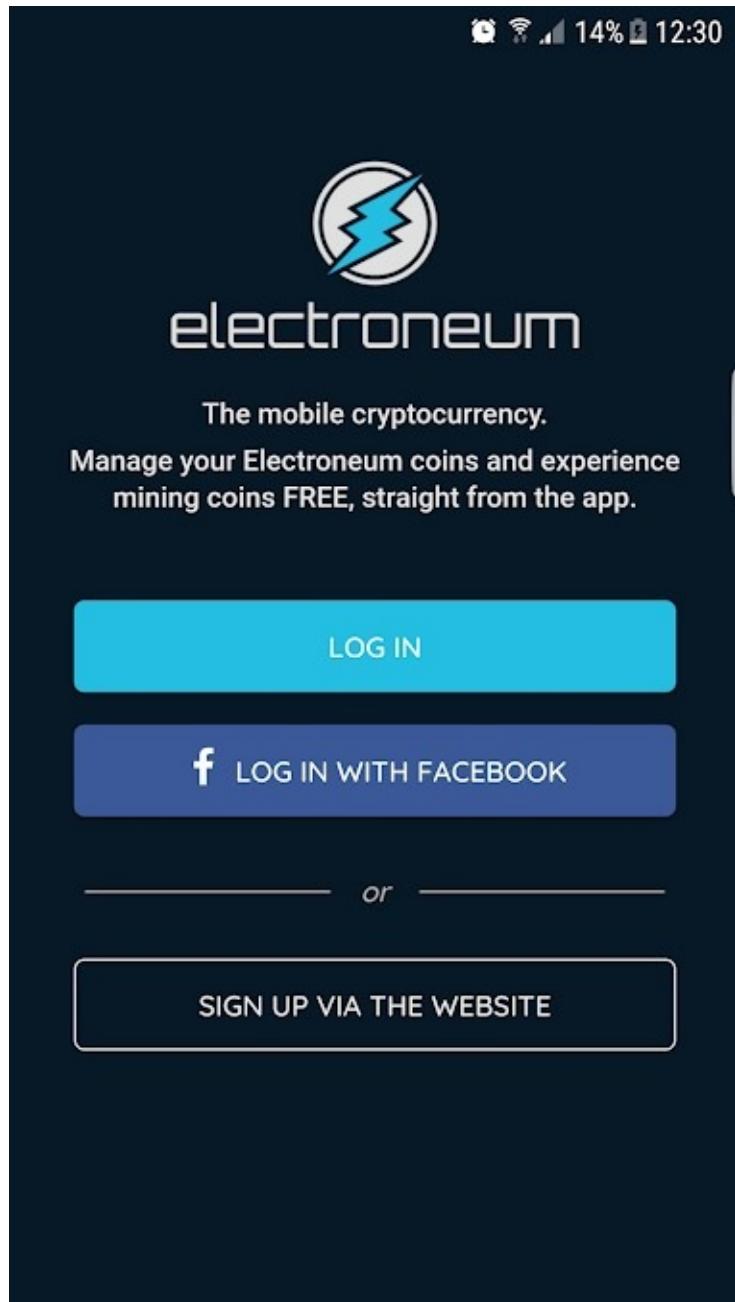
- Version Android : <https://play.google.com/store/apps/details?id=com.electroneum.mobile>

Dans ce TP, nous allons, à l'aide des composants tenter de nous rapprocher le plus possible de l'application (voir screen en pièce jointe)

1) Créez une nouvelle page que vous nommerez Login. Ce sera notre page de connexion et d'inscription.

Faites que cette page soit la page par défaut, en modifiant le fichier **src/app/app.component.ts**

2) Utilisez les composants Ionic pour rapprocher cette page le plus proche possible du screen ci-dessous, en adaptant au passage le style et le texte.



De plus, vous rajouterez juste avant le bouton de login deux input pour saisir son identifiant et mot de passe.

3) Faites qu'au clic sur le bouton login on puisse accéder à la page d'accueil avec les différents onglets (voir pour inspiration la fonction gotoHome() :

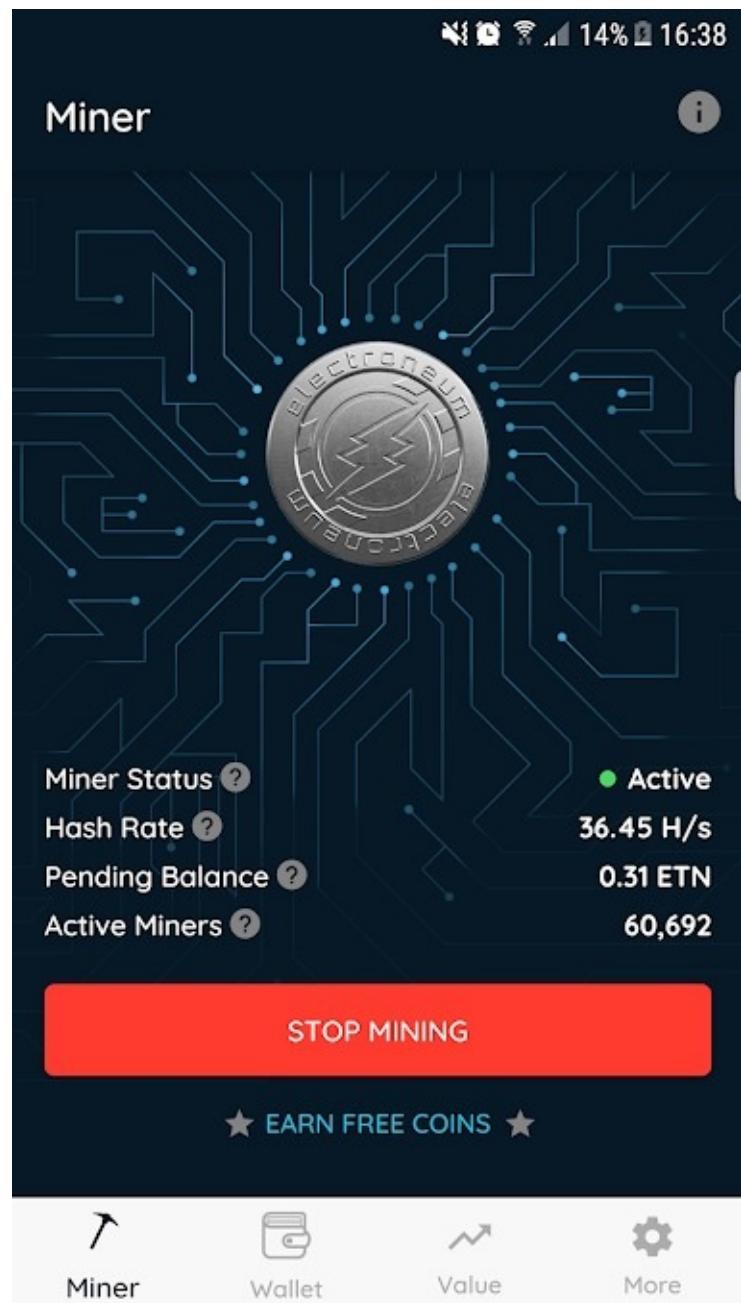
<https://github.com/charlesen/duckcoin/blob/master/src/pages/profile/profile.ts>)

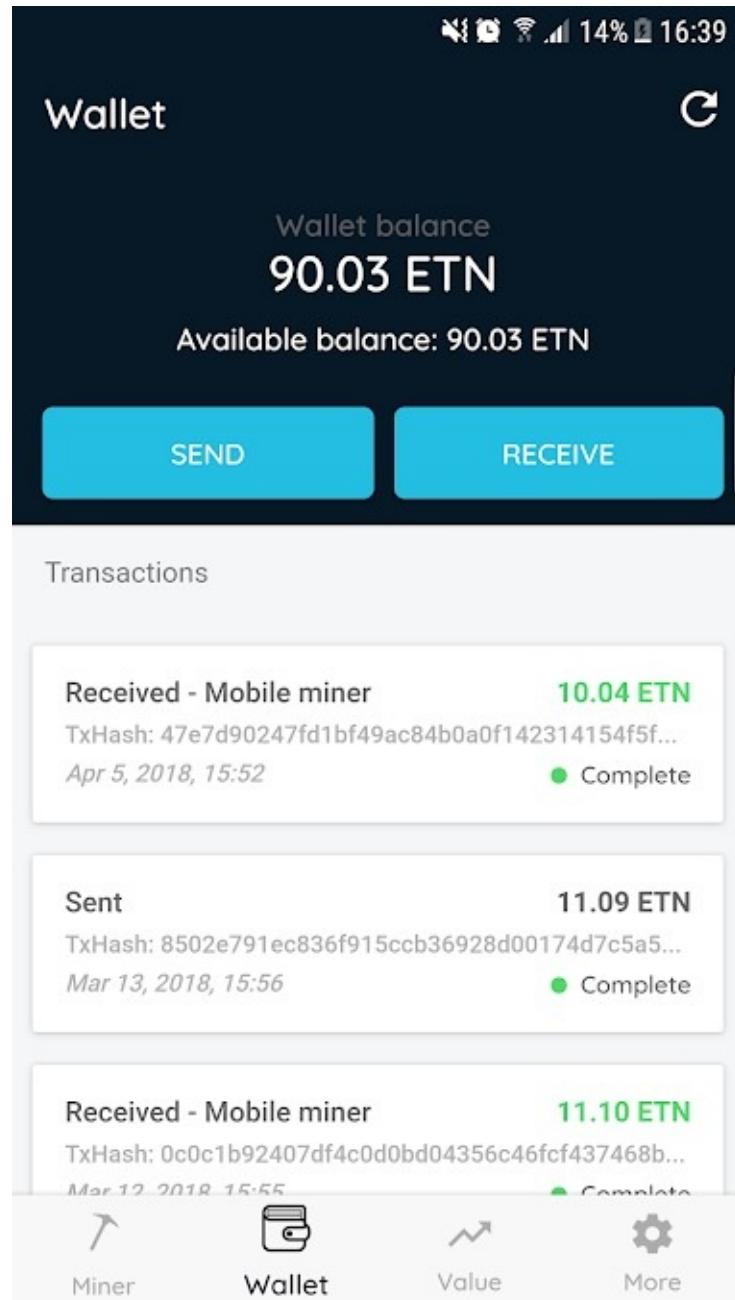
4) Sans modifier l'onglet Accueil, adaptez les autres onglets Minage, Portefeuille

P.S. : Ionic propose un ensemble d'icônes que l'on peut intégrer facilement dans l'appli.

```
<ion-icon name="home"></ion-icon>
```

La liste de toutes les icônes disponibles se trouve à l'adresse <https://ionicframework.com/docs/ionicons/>





5) Améliorez les onglets Profil et More (page Settings créé dans le TP précédent) avec différents autres composants. Dans l'onglet Setting on pourra par exemple avoir la possibilité de choisir plusieurs devises différentes et d'en cocher une par défaut, ou encore de choisir la langue par défaut,...laissez libre cours à votre imagination ;-)

Annexes

- Liste des composants Ionic et Documentation : <https://ionicframework.com/docs/components/>

Chap 6 - Introduction au langage TypeScript, le futur de JavaScript

TypeScript est un langage de programmation libre et open source développé par Microsoft qui a pour but d'améliorer et de sécuriser la production de code JavaScript. Sortie en 2012, Il est vu par beaucoup comme le futur du Javascript, car se basant en sur la norme ECMAScript 6, celle déjà intégré au moteur JavaScript de la plupart des navigateurs et qui fera foi dans les prochaines années.

TypeScript c'est donc du JavaScript, avec de supers pouvoirs, utilisé par la plupart des Frameworks JavaScript du moment.

React
If `create-react-app` had an option to include TypeScript, this is what you'd get: the perfect starting point for React + TypeScript.

React, Redux, Webpack, Enzyme, Jest

Angular
The Angular team recommends using TypeScript with all your Angular apps. Check out their great tutorial on getting started with both.

Angular, Protractor, Jasmine, Karma, Rollup, Uglify, TSLint

Express
Check out this starter project using Express + MongoDB + TypeScript and see why TypeScript and Node.js are a perfect combination.

Node.js, Express, Pug, Jest, TSLint, MongoDB, Travis CI

Babel
Leverage type-checking with TypeScript in your existing Babel projects using Babel 7 and later.

Babel

React Native
Take your JavaScript to mobile devices with React Native while keeping your productivity intact with TypeScript.

React Native, Enzyme, Jest

Vue.js
Check out this TodoMVC app using Vue.js and TypeScript.

Vue.js, Webpack

Les fichiers définis dans ce langage ont pour extension **.ts**. Les navigateurs web ne sachant pas encore interpréter du code en TypeScript pur, il est nécessaire de le compiler en JavaScript : on parle alors de transtypage.

Installation

Ionic utilise une version "interne" de TypeScript, mais pour nos tests nous allons devoir installer le package de manière globale.

```
> npm install -g typescript
```

Un premier script

Créons un fichier à la racine du projet **demo_typescript.ts** et tant qu'à faire ajoutons-y du contenu :

demo_typescript.ts

```
function ditBonjour(person) {
  return "Bonjour, " + person;
}

let user = "Raphael";

document.body.innerHTML = ditBonjour(user);
```

Il faut ensuite, compiler ce code pour générer un fichier .js interprétable par le navigateur.

```
$ tsc demo_typescript.ts
$ ls -l
total 304
-rw-rw-r--  1 charles charles  6180 avril 11 22:14 config.xml
-rw-rw-r--  1 charles charles   131 avril 13 07:12 demo_typescript.js
-rw-rw-r--  1 charles charles   133 avril 13 07:04 demo_typescript.ts
```

demo_typescript.js

```
function ditBonjour(person) {
  return "Bonjour, " + person;
}
var user = "Raphael";
document.body.innerHTML = ditBonjour(user);
```

Bien évidemment le contenu du fichier **demo_typescript.js** est strictement le même que celui du fichier **.ts**, car ce code est relativement simple.

Mais tout l'intérêt de TypeScript est surtout de pouvoir ajouter à JavaScript des notions de classes, d'interface, d'héritage ou de Polymorphisme,...

Créons une classe Licence et ajoutons-y des paramètres et quelques méthodes au passage.

Types de base

Contrairement à JavaScript où les types sont définis au remplissage d'une variable, TypeScript propose un typage de variable beaucoup plus fort.

La définition générale d'une variable se fait de la manière suivante :

```
let nomDeLaVariable: leTypeDeBase [= valeur par défaut - Optionnel];
```

Booléens

```
let isConnected: boolean = false;
```

Nombres

```
let valeur_decimal: number = 6;
let valeur_hex: number = 0xf00d;
let valeur_binary: number = 0b1010;
let valeur_octal: number = 0o744;
```

String

```
let color: string = "blue";
color = 'red'; # On a le choix entre les guillemets simples ou doubles
```

Chose intéressante pour les chaînes de caractères, il est possible de les utiliser sous forme de template, un truc que l'on rencontrait jusqu'à lors dans des langages de haut niveau comme Python.

```
let fullName: string = `Charles EDOU NZE`;
let age: number = 30;
let sentence: string = `Salut, mon name est ${ fullName }.
J'aurai ${ age + 1 } ans à la fin de l'année. `;
```

Ceci est l'équivalent de concaténer des chaines de caractères avec le signe "+".

Les tableaux

```
let list_nombres_premiers: number[] = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31];
```

Any

On l'utilise quand on ne sait pas trop qu'elle type de données on aura à traiter. C'est souvent le cas quand on utilise un API propriétaire.

```
let variableApi: any = 4;
variableApi = "c'est une chaîne de caractère";
variableApi = false; // ou un booléan finalement
```

Les Classes et les méthodes

Un des gros avantages de TypeScript est la possibilité de créer des classes, ce qui n'était pas tout à fait (au sens strict) en JavaScript, même s'il était possible de créer un genre de classe avec l'attribut prototype.

Les méthodes elles sont équivalentes aux fonctions JavaScript, mais le mot clé "function".

```
class classeMmi {
    intervenant: string;
    etudiants: string[];
    constructor(intervenant: string, etudiants:string[]) {
        this.intervenant = intervenant;
        this.etudiants = etudiants;
    }
    getIntervenant() {
        return "L'intervenant actuel s'appelle " + this.intervenant;
    }
    getEtudiants() {
        return this.etudiants;
    }
}
```

```

    }

let classeLPMim = new classeMmi("charles", ['etudiant1','etudiant_n']);
classeLPMim.getIntervenant() // Retourne L'intervenant actuel s'appelle charles

```

Les itérateurs

Boucle for..of

Cette boucle permet d'afficher les éléments d'une liste pure.

```

let unArray = [1, "deux", 'trois'];

for (let valeur of unArray) {
    console.log(valeur); // 1, "deux", "trois"
}

```

cette boucle est aussi celle qu'il faudra utiliser quand on voudra boucler sur des listes d'objets :

```

let unArrayObjets = [{nom:'etudiant 1', num:1}, {'nom':'etudiant 2', 'num':2}];

for (let item of unArrayObjets) {
    console.log(item.nom); // etudiant 1 etudiant 2
}

```

Boucle for..in

Reprenons le code précédent, mais en utilisant la boucle for..in

```

let unArray = [1, "deux", 'trois'];

for (let index in unArray) {
    console.log(index); // 0, 1, 2
}

```

La boucle renvoie en fait les indexes de chaque valeur et ne doit être utilisé dans ce cas là que si l'on souhaite vraiment récupérer les indexes d'un tableau.

Mais le véritable intérêt pour moi de la boucle **for..in** est de pouvoir boucler sur les propriétés d'un objet.

```

let etudiant = {'nom':'Etudiant 001', 'num':1};
for (let cle in etudiant) {
    console.log(etudiant[cle]) // Renvoie Etudiant 001
}

```

Et un petit résumé de ce que renvoie l'une ou l'autre des deux boucles :

```

let list = [4, 5, 6];
let etudiant = {'nom':'Etudiant', 'num':1};

for (let i in list) {
    console.log(i); // "0", "1", "2",
}

for (let i of list) {
    console.log(i); // "4", "5", "6"
}

```

Calculs et transformations sur les listes avec map, filter et reduce

Map

Considérons la liste de politiciens suivante :

```
let politiciens = [
  {
    'prenom': 'Emmanuel',
    'nom': 'Macron',
    'age': 40,
  },
  {
    'prenom': 'Edouard',
    'nom': 'Philippe',
    'age': 47,
  },
  {
    'prenom': 'Bruno',
    'nom': 'Le Maire',
    'age': 49,
  },
  {
    'prenom': 'Virginie',
    'nom': 'Calmels',
    'age': 47,
  },
  {
    'prenom': 'Alain',
    'nom': 'Juppé',
    'age': 72,
  },
];
```

Pour extraire uniquement les noms tout en majuscule de chacun d'entre eux, JavaScript propose différentes solutions comme celle-ci :

```
let politiciens_noms = [];

for (let i = 0, max = politiciens.length; i < max; i += 1) {
  politiciens_noms.push(politiciens[i].nom.toUpperCase());
}

// politiciens_noms
// (5) ["MACRON", "PHILIPPE", "LE MAIRE", "CALMELS", "JUPPÉ"]
```

Ou encore celle là :

```
let politiciens_noms = [];

politiciens.forEach(function (politicien) {
  politiciens_noms.push(politicien.nom.toUpperCase());
});
```

Mais avec la méthode Map, la récupération se fait de manière beaucoup plus efficace :

```
let politiciens_noms = politiciens.map(function (politicien, index, array) {
```

```

    return politicien.nom.toUpperCase();

});
// politiciens_noms
// (5) ["MACRON", "PHILIPPE", "LE MAIRE", "CALMELS", "JUPPÉ"]

```

La méthode donne accès dans son callback à chaque item du tableau depuis la variable **politicien**, à sa position (**index**) à l'intérieur du tableau, et enfin du tableau lui-même (**array**).

Avec cette méthode, vous n'aurez pas à vous inquiéter de l'index de la boucle ou d'utiliser la méthode `push` pour stocker vos éléments. De plus, la méthode renvoyant un Array, il est tout à fait possible de d'appliquer une autre méthode juste après la méthode `map`.

Filter

Cette méthode fait exactement ce que son nom semble suggérer : à partir d'un tableau reçu en entrée, il le filtre en éliminant les éléments non désiré selon une condition déterminée.

Reprenons notre liste de politiciens et ne retenons que ceux de moins de 50 ans. Grâce à la méthode `filter`, il suffit de faire :

```

let politiciens_U50 = politiciens.filter((politicien) => politicien.age <= 50 );

// (4) [...], [...], [...], [...]
// 0:{prenom: "Emmanuel", nom: "Macron", age: 40}
// 1:{prenom: "Edouard", nom: "Philippe", age: 47}
// 2:{prenom: "Bruno", nom: "Le Maire", age: 49}
// 3:{prenom: "Virginie", nom: "Calmels", age: 47}
// length:4
// __proto__:Array(0)

```

Reduce

Si la fonction `map` permet de créer un nouveau tableau en transformant chaque élément d'un tableau, et si `filter` permet quant à lui de créer un nouveau tableau en supprimer des éléments selon une condition déterminée, la méthode `reduce` permet de prendre tous les éléments du tableau pour les "réduire" à une unique valeur.

Considérons le tableau de politiciens de moins de 50 ans précédent. On aimerait cette fois calculer la moyenne d'âge de ces "*jeunes*" faiseurs de lois.

```

let moyenne_age = politiciens_U50.reduce(function (previous, current, index) {
    return (previous + current.age);
}, 0)/politiciens_U50.length;

// previous est la valeur de l'âge à un moment t dans la boucle. Cette valeur est initialisé à 0.
// current est la valeur actuelle de l'élément de la boucle, ex: {prenom: "Emmanuel", nom: "Macron", age: 40}

// moyenne_age : 45.75

```

Vous aurez souvent l'occasion d'utiliser ces méthodes qui sont extrêmement puissante. On peut comme dit précédemment les concaténer les unes derrière les autres. On peut par exemple calculer la moyenne d'âge des politiciens de moins 50 ans précédents en une seule ligne de code. Oui je sais, je suis cruel de vous l'avoir caché jusque là :-).

```

politiciens.filter((politicien) => politicien.age <= 50 )
    .reduce(function (previous, current, index) {
        return previous + current.age;
    })

```

```

        }, 0)/politiciens.filter((politicien) => politicien.age <= 50 )
.length;

// Renvoie bien : 45.75

```

Les conditions

Comme en JavaScript. On retrouve les traditionnels "if..else" , "switch...case" :

```

if(une_conditon) {
    // La condition est vraie
} else {
    // Elle est fausse
}

```

Et un switch-case :

```

switch(meteo) {
    case 'soleil': {
        //Il fait beau
        break;
    }
    case 'pluie': {
        // Il fait moins beau
        break;
    }
    default: {
        // devine
        break;
    }
}

```

Constantes

Une constante comme son nom le suppose est censé de ne pas être changé par la suite ou redéfini par la suite. On pourra par exemple y stocker une URL d'une API.

```
const apiUrl = 'https://duckcoin.charlesen.fr';
```

Comment TypeScript s'intègre à Ionic

En fait, TypeScript est présent partout ou presque dans Ionic. Les classes définies dans les fichiers .ts de chaque page sont écrites, comme vous vous en doutiez sûrement, en TypeScript :

```

export class HomePage {
    selected : any = '';
    items : any = [];
    constructor(public navCtrl: NavController) {
        this.items = [
            {'title':'Bitcoin', 'currency':'btc', 'price':'5000€'},
            {'title':'Ethereum', 'currency':'eth', 'price':'500€'},
            {'title':'Ripple', 'currency':'xrp', 'price':'0.4€'}
        ];
    }

    itemSelected(item) {
        this.selected = item;
    }
}

```

```

    }
}

}

```

Exercez-vous

- 1) On veut afficher la liste des dernières transactions de la Blockchain sous forme de liste dans l'onglet Accueil, tout en bas du texte de bienvenue. On définit la liste de transactions ci-dessous :

```

this.transactions = [
{
  'sender': 'charles',
  'recipient': 'maxime',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'raphael',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'doreen',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'louis-joseph',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'elise',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'germain',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'anthony',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'pol',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'vincent',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'nicolas',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'kevin',
  'amount': 100,
},

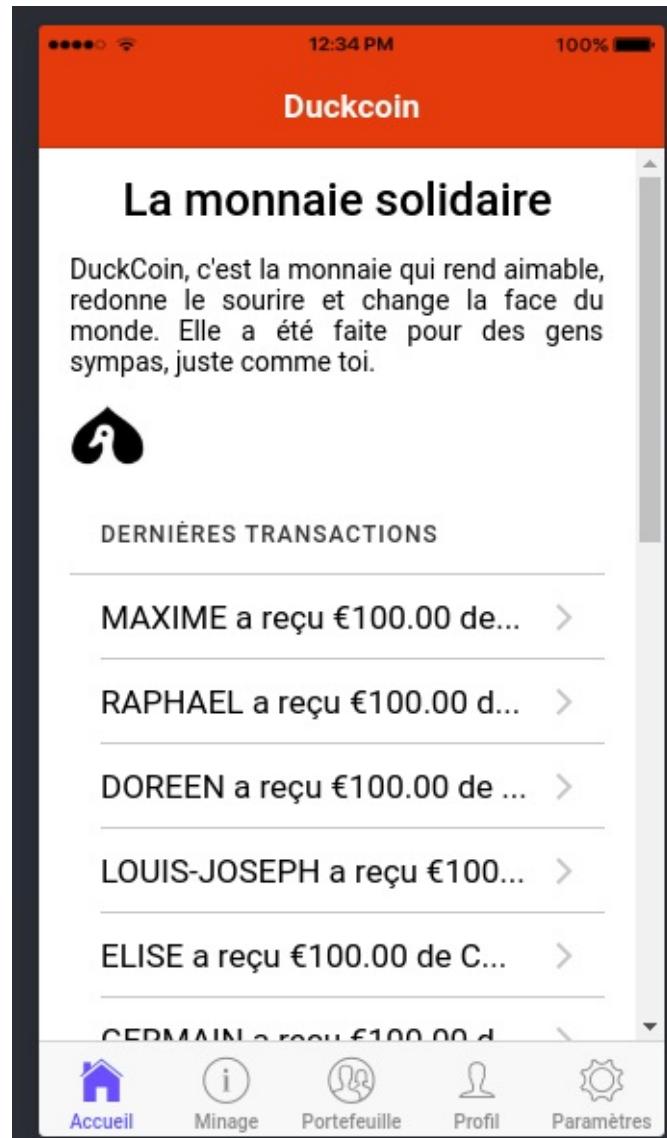
```

```
{
  'sender': 'charles',
  'recipient': 'willy',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'elodie',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'adrien',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'romain',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'quentin',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'jean-etienne',
  'amount': 100,
},
{
  'sender': 'charles',
  'recipient': 'gael',
  'amount': 100,
}
];
```

En vous basant sur ce qui a été fait précédemment avec la liste des cryptomonnaies Bitcoin, Ethereum et Ripple, faites en sorte que l'on puisse visualiser la liste des dernières transactions en Page d'accueil.



3) Le signe dollar (\$) de la liste est ajouté automatiquement grâce à un pipe Angular (concept abordé au [chapitre 7](#)). Comment d'après la documentation suivante, est-il possible de remplacer le \$ en € ? Puis € en DRT ?
<https://angular.io/api/common/CurrencyPipe>.



2) Faites qu'en cliquant sur une ligne de la transaction, on affiche une fenêtre modale, avec le résumé de la transaction :

<https://ionicframework.com/docs/components/#modals>

P.S : il est possible de passer des paramètres à une fenêtre modale, puis de les récupérer dans la fenêtre concernée :

Ouverture de la fenêtre avec passage de paramètres :

```
import { ModalController } from 'ionic-angular';
import { MaPageModal } from './modal/modal'; // la page modale est dans le même dossier que la principale

export class MaPage {
  constructor(public modalCtrl: ModalController) {}

  presentModal() {
    let modal = this.modalCtrl.create(ModalPage, {'monParam':'Ceci est un paramètre'});
    modal.present();
  }
}
```

Récupération des données dans la classe de la fenêtre :

```
export class MaPageModal {
  unParamVenuDeLoin : any;

  constructor(
    public platform: Platform,
    public params: NavParams,
    public navCtrl: ViewController
  ) {
    this.unParamVenuDeLoin = this.params.get('monParam')
  }

  dismiss() {
    this.navCtrl.dismiss();
  }
}
```

Annexes

- Documentation sur les méthodes Map, filter et Reduce :
 - <https://code.tutsplus.com/tutorials/how-to-use-map-filter-reduce-in-javascript--cms-2620>
 - <https://scotch.io/tutorials/list-processing-with-map-filter-and-reduce>

Chap 7 - Introduction au framework Angular



AngularJS est un framework JavaScript libre et open source développé par Google. Il permet la construction d'applications client réactives en HTML et en TypeScript. Angular est d'ailleurs lui-même écrit en TypeScript.

Angular, propose un outil, Angular CLI (Command line), qui facilite la création et le développement de projets web en ligne de commande.

Pour mieux comprendre le fonctionnement d'Angular, rien de mieux que développer un petit projet web basé sur celui-ci.

Installation d'Angular CLI

```
$ npm install -g @angular/cli # Rajouter "sudo" si nécessaire
```

Création d'un nouveau projet

On va se mettre en dehors de notre projet DuckCoin, pour éviter de le polluer. Dans un dossier autre que celui de Duckcoin, saisir la commande suivante :

```
$ ng new duckweb # Cette commande va créer un nouveau dossier duckweb.
```

Ca va mettre un peu de temps à se créer, mais pas de panique vous êtes sur la bonne voie ;-).

```
~/www/lab/duckweb$ ll
total 484
drwxrwxr-x  2          4096 avril 13 14:41 e2e
-rw-rw-r--  1          923  avril 13 14:41 karma.conf.js
drwxrwxr-x 891        32768 avril 13 14:43 node_modules
-rw-rw-r--  1         1292  avril 13 14:41 package.json
-rw-rw-r--  1        423000 avril 13 14:43 package-lock.json
-rw-rw-r--  1          722  avril 13 14:41 protractor.conf.js
-rw-rw-r--  1         1023  avril 13 14:41 README.md
drwxrwxr-x  5          4096 avril 13 14:41 src
-rw-rw-r--  1          363  avril 13 14:41 tsconfig.json
-rw-rw-r--  1         3012  avril 13 14:41 tslint.json
```

Une fois la création terminée, on va pouvoir lancer notre projet :

```
$ cd duckweb
$ ng serve --open
```

Welcome to app!



En avant d'aller plus loin découvrons un peu la structure, puis l'architecture d'un projet Angular.

Structure d'un projet Angular

À l'intérieur d'un projet angular on trouve un certain nombre de dossiers et de fichiers :

```

drwxrwxr-x  2          4096 avril 13 14:41 e2e
-rw-rw-r--  1          923  avril 13 14:41 karma.conf.js
drwxrwxr-x  891        32768 avril 13 14:43 node_modules
-rw-rw-r--  1          1292  avril 13 14:41 package.json
-rw-rw-r--  1          23000  avril 13 14:43 package-lock.json
-rw-rw-r--  1          722   avril 13 14:41 protractor.conf.js
-rw-rw-r--  1          1023  avril 13 14:41 README.md
drwxrwxr-x  5          4096 avril 13 14:41 src
-rw-rw-r--  1          363   avril 13 14:41 tsconfig.json
-rw-rw-r--  1          3012  avril 13 14:41 tslint.json

```

- **e2e** : ce dossier stock des scripts pour effectuer des tests unitaires, un ensemble de d'énoncé et d'instruction qui permettent de vérifier que son code fonctionne bien selon un certain cahier des charges.
- **node_modules** : c'est dans ce dossier que sont installés tous les plugins Node installé via npm.
- **src** : c'est dans ce dossier seront stockés nos fichiers sources, le code quoi. C'est dans ce dossier que l'on passera 99% du temps.
- **angular-cli.json** : un fichier de configuration pour Angular CLI.
- **package.json** : fichier de configuration pour Node
- **protractor.conf.js** : Protractor est un outil utilisé pour les Tests unitaires. Ce fichier de configuration est utilisé par lui.
- **karma.conf.js** : karma est un autre outil utilisé dans les tests unitaires. Tester son projet est une philosophie forte chez Angular.
- **tsconfig.json** : fichier de configuration pour le compilateur de TypeScript (tsc).
- **tslint.json** : tslint est utilitaire qui permet de vérifier les fichiers TypeScript (bug, import non utilisé,...)

Architecture d'un projet Angular

Le bloc de base d'une application Angular est le composant, qui peut être vu comme la combinaison :

- D'une **Vue** : du contenu HTML
- D'un **Modèle** de données : les informations qui vont être affichées dans le contenu HTML
- D'un **Contrôleur**, qui va se charger de la logique derrière l'affichage des données dans la vue.

Un composant peut être constitué d'autres composants. Par exemple :

- **Twitter** [Composant Root]
 - **Entête** (Titre, logo,...)
 - **Un contenu** principal [Composant Content]
 - **Tweets** [Composant liste de Tweets]
 - **Un tweet** [Composant Tweet] est constitué de contenu
 - ce contenu peut être soit une image [Composant image], soit du texte [Composant texte]
 - ce contenu est aussi fait de commentaires [Composant Commentaire]

L'intérêt d'une architecture en composants est que si jamais on souhaite étendre une fonctionnalité particulière, plutôt que de la redefinir, on va créer un composant qui pourra être appelé partout (afficher des tweets en page d'accueil, sur son profil, dans les résultats de recherche,...).

Le composant principal d'Angular est défini à l'intérieur du fichier **src/app/app.component.ts**, où l'on retrouve aussi d'autres fichiers qui forment le MVC du projet.

```

app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts

```

Une application a toujours au moins un module racine qui permet le lancement du projet (à l'exemple d'un fichier index.html en racine d'un site web). C'est ce module qui va amorcer le composant Root (**AppComponent**).

Par convention, celui-ci s'appelle **AppModule** et est défini dans le fichier **src/app/app.module.ts**.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
    TransactionComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

S'il fallait faire un comparatif avec un véhicule, les composants seraient des éléments comme le pare-brise, les retroviseurs, les roues, ...tandis que le module Root serait le moteur, sans lequel le véhicule, même le plus beau au monde (avec les plus beaux composants), ne pourrait démarrer.

Angular dans les templates

*ngFor

Permet de boucler sur les éléments d'un tableau à l'intérieur d'un template html.

Supposons que l'on ait défini la liste des mois de l'année dans une liste :

```
let months_of_year = ['Janvier', 'Fevrier', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet',
                      'Aout', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
```

l'affichage de tous les éléments se fait simplement de la manière suivante :

```
<div *ngFor="let month of months_of_year">
  Mois de l'année : {{month}}
</div>
```

*ngIf

Comme vous pouvez le deviner, *ngIf est le **"if...else"** adapté aux templates.

```
<span *ngIf=" isConnected">Je suis connecté.</span>
```

Pipe

Comme dans la plupart des moteurs de templates, Angular permet l'utilisation de pipes qui permettent de modifier une variable ou un contenu avant qu'il soit affiché. Le framework propose un certain nombre de pipes prêts à l'emploi, comme *titlecase*, *currency*,...Mais il est tout à fait possible de créer son propre pipe.

```
<div>
  <h2>{{ 'charles edou nze' | titlecase }}</h2>
</div>
```

titlecase permet de mettre en capitale les premières lettres de chaque mot. Ce qui donnera le résultat suivant :

```
Charles Edou Nze
```

Exercez-vous

- 1) Créez un nouveau projet comme expliqué précédemment
- 2) Que pouvez-vous remarquer dans l'arborescence de fichiers d'Angular...vis-à-vis de Ionic
- 3) A l'aide de la commande suivante, générez un nouveau Composant nommé "**transaction**" :

```
$ ng g c transaction
```

Que s'est-il passé ? Ouvrez le fichier **src/app/transaction/transaction.component.ts** et examinez-le.

- 4) Ouvrez le fichier **src/app/app.component.html**, et remplacez le contenu ci-dessous (on ne gardera que le logo d'Angular)

```
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://github.com/angular/angular-cli/wiki">CLI Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>
```

Par le code suivant :

```
<app-transaction></app-transaction>
```

Que s'est-il passé dans ce qui est affiché dans votre navigateur ?

P.S : notre application a désormais l'architecture suivante :

- **DuckWeb**
 - **Transaction**

- 5) Remplacez le contenu du fichier **src/app/transaction/transaction.component.html**. Quel changement dans votre navigateur ?

- 6) Ouvrez le fichier **src/app/transaction/transaction.component.ts** dans le constructeur, définissez une liste de transactions comme vu dans le TP précédent, puis modifiez le fichier **src/app/transaction/transaction.component.html**, de manière à avoir le résultat suivant :

Welcome to app!



Liste des transactions

MAXIME a reçu €100.00 de CHARLES
RAPHAEL a reçu €100.00 de CHARLES
DOREEN a reçu €100.00 de CHARLES
LOUIS-JOSEPH a reçu €100.00 de CHARLES
ELISE a reçu €100.00 de CHARLES
GERMAIN a reçu €100.00 de CHARLES
ANTHONY a reçu €100.00 de CHARLES

Que pouvez-vous conclure sur le rôle d'un composant ? Comprenez-vous mieux comment fonctionne les composants Ionic ?

- 7) Dans le fichier **src/app/transaction/transaction.component.ts**, ajoutez les lignes suivantes dans la partie dédiée aux imports :

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

Puis modifiez le fichier comme ceci :

```
import { Component, OnInit } from '@angular/core';
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

const apiUrl='https://duckcoin.charlesen.fr';

@Component({
  selector: 'app-transaction',
  templateUrl: './transaction.component.html',
  styleUrls: ['./transaction.component.css']
```

```

})
export class TransactionComponent implements OnInit {
  transactions: any = [];
  constructor(private http: HttpClient) {
    // On récupère du contenu via une requête HTTP GET
    this.http.get(`${apiUrl}/transactions`).subscribe(
      data => {
        this.transactions = data['transactions'];
      }, err => {
        console.log("Une erreur s'est produite.")
      });
    //...
  }

  ngOnInit() {
  }

}

```

Vous avez planté normalement. Savez-vous pourquoi ?

```
No provider for HttpClient!
```

Dans le fichier **src/app/app.module.ts** ajoutez les lignes suivantes :

```

import { HttpClientModule } from '@angular/common/http';
// ...
imports: [
  BrowserModule,
  HttpClientModule,
  // ...
  IonicModule.forRoot(MyApp, {
    // tabsPlacement: 'top',
    backButtonText: 'Retour'
  })
],
...

```

Grâce à l'utilisation du module http, vous pouvez facilement récupérer des données directement depuis le site hébergeant la blockchain.

Vous retrouverez les sources, ainsi que les corrections des exercices de ce TP à l'adresse:

<https://github.com/charlesen/ionic-book/tree/master/examples/duckweb>

Annexes

- Angular Tutorial : <https://www.tutorialspoint.com/angular4/index.htm>
- Documentation officielle :
 - <https://angular.io/docs>
 - <https://cli.angular.io/>

Chap 8 - Architecture avancée d'une application Ionic : Composants, Directives, Providers, Services, Pipes, Modules, persistance de données et plugins natifs

Composants

La majorité des développements sous Angular, et comme vous l'aurez compris, sous Ionic également (Ionic étant en fin de compte un projet Angular), est effectué au niveau des composants. Nous avons déjà étudier le composant Root, dont l'arborescence est la suivante :

```
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
```

Nous avons aussi écrit un composant **Transaction** qui nous permettait d'afficher la liste de nos dernières transactions.

Welcome to app!



Liste des transactions

MAXIME a reçu €100.00 de CHARLES
RAPHAEL a reçu €100.00 de CHARLES
DOREEN a reçu €100.00 de CHARLES
LOUIS-JOSEPH a reçu €100.00 de CHARLES
ELISE a reçu €100.00 de CHARLES
GERMAIN a reçu €100.00 de CHARLES
ANTHONY a reçu €100.00 de CHARLES

La création d'un nouveau composant se fait simplement en saisissant la commande suivante à la racine de votre projet ionic :

```
$ ionic g component monComposant
[OK] Generated a component named monComposant!

src/components/moncomposant.css
src/components/moncomposant.html
src/components/moncomposant.spec.ts
src/components/moncomposant.ts
src/components/components.module.ts
```

On voit ici qu'un module (components.module.ts) a aussi été créé. Rappelons que les modules sont chargés du bootstrapping (démarrage) d'un composant. C'est donc ce module qu'il faudra déclarer dans le module principal **src/app/app.module.ts** :

```
// Modules
import { ComponentsModule } from './components/components.module';

// ...

imports: [
  BrowserModule,
  HttpClientModule,
  ComponentsModule, // Importer le module ici
  IonicModule.forRoot(MyApp, {
    // tabsPlacement: 'top',
    backButtonText: 'Retour'
  })
],
// ...
```

Il faut également modifier le fichier **src/components/components.module.ts** comme ceci :

```
import { NgModule } from '@angular/core';
import { IonicModule } from 'ionic-angular'; // On rajoute cette ligne
import { MonComposantComponent } from './components/components';
@NgModule({
  declarations: [MonComposantComponent],
  imports: [IonicModule], // ...Et celle-ci
  exports: [MonComposantComponent]
})
export class ComponentsModule {}
```

Je vous rassure, vous n'aurez pas à faire tout cela à chaque création de composant, en fait l'intérêt de regrouper tout cela dans un module permet de créer de nouveaux composants, sans devoir les redéclarer dans toute l'application.

A présent vous pouvez appeler votre composant sous forme de tag dans n'importe quel fichier html de l'application.

```
<moncomposant></moncomposant>
```

Directives

Une directive est un élément qui va nous permettre d'étendre des fonctionnalités html. Il en existe différents types :

- **Directive de type attribut** : vous en avez déjà vu, elles permettent de modifier du html. Citons par exemple **text-center**, une directive qui permet de centrer le contenu d'un élément, ou encore la directive **padding**, qui

permet d'ajouter un padding à l'élément qui l'invoque.

- **Directive de type composant** : oui au risque de vous embrouiller un peu, un composant est en réalité une directive, mais dotée d'un template html. La directive est en quelque sorte l'atome, le composant la molécule.
- **Directive de type structure** : Ce type de directive sont faite pour la manipulation du DOM et commencent toujours par un **“*”**. On peut citer parmi celles que nous avons déjà utilisé les directives ***ngIf** et ***ngFor**.

la création d'une directive se fait simplement en saisissant la commande suivante :

```
$ ionic g directive maDirective
[OK] Generated a directive named maDirective!
```

Créons par exemple une directive que nous appelerons **bolder** et qui permettra de mettre en gras l'élément qui l'appelerait.

```
$ ionic g directive bolder
[OK] Generated a directive named bolder
```

On déclare ensuite une fois pour toute le "module mère" de toutes les directives dans le module root **src/app/app.module.ts** :

```
// ...
//Modules
import {ComponentsModule} from './components/components.module';
import {DirectivesModule} from './directives/directives.module';
import { HttpClientModule } from '@angular/common/http';

//...

imports: [
  BrowserModule,
  HttpClientModule,
  ComponentsModule,
  DirectivesModule, // ICI
  IonicModule.forRoot(DuckCoinApp, {
    // tabsPlacement: 'top',
    backButtonText: 'Retour'
  })
],
```

Puis, on édite notre directive pour qu'il fasse ce que l'on souhaite, à savoir mettre du contenu en gras :

```
import { Directive, ElementRef } from '@angular/core';

/**
 * Directives.
 */
@Directive({
  selector: '[bolder]' // Attribute selector
})
export class BolderDirective {

  constructor(Element: ElementRef) {
    Element.nativeElement.style.fontWeight = 'bolder';
  }
}
```

Il ne nous reste plus qu'à utiliser notre nouvelle directive sur du contenu en page d'accueil par exemple :

```
<span bolder>mon texte en gras</span>
```

Services ou Providers

Comme nous l'avons vu, un composant permettant d'afficher du contenu à plusieurs endroits à partir d'un code unique. C'est le cas par exemple du composant **<transaction></transaction>** que l'on peut appeler dans n'importe quelle template html de notre application.

Supposons que l'on souhaite récupérer la liste des dernières transactions sous forme de tableau comme c'est le cas dans la classe **TransactionComponent** :

```
export class TransactionComponent {

  transactions: any[] = [];

  constructor(private http: HttpClient) {
    this.http.get(`apiUrl/transactions`).subscribe(
      data => {
```

```
    this.transactions = data['transactions'];
  }, err => {
  console.log("Error occurred.")
});
}

}
```

Une première solution serait de copier le code de cette classe. Mais si une autre page souhaite également avoir accès à cette même liste, la copie alors apparaît comme une mauvaise solution.

C'est là qu'interviennent les **Services**, qui sont en fait des bouts de codes métiers, des méthodes, qui peuvent appeler dans d'autres pages, sans devoir les réécrire. On code une fois, on les réutilise partout.

Nous aurons par exemple besoin des services pour la gestion des sessions utilisateurs. En effet, à peu près toutes les pages de notre application auront besoin de s'assurer que notre utilisateur courant est bien connecté.

```
$ ionic g provider User
[OK] Generated a provider named User!
```

Cette commande va créer un nouveau service **User**, dans lequel nous déclarerons un certain nombre de méthodes pour la gestion de l'authentification, la création de comptes utilisateur,...

Pipes

On en a déjà un peu parlé au chapitre. Les pipes permettent de modifier la forme d'un contenu avant son affichage. Citons quelques pipes intéressants :

- **currency** : permet de rajouter une devise avant la valeur sur laquelle on l'applique
- **date** : formatage de date
- **uppercase** : transforme du texte en majuscule
- **lowercase** : transforme du texte en minuscule
- **json** : affiche le contenu d'un objet ou d'un texte au format JSON
- ...

```
let maVariable = "Hello mmi";
...
<span>{{maVariable | uppercase}}</span>
```

Pour plus de détails : https://www.tutorialspoint.com/angular4/angular4_pipes.htm

Modules

Un module angular permet de regrouper en un seul endroit des composants, des directives, des pipes et des services de l'application. Vous n'aurez pas nécessairement besoin de créer des modules manuellement, car la création par exemple d'un composant le fera pour vous.

Prenons par exemple le module racine de notre application défini comme ceci :

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
```

```
  AppComponent
],
imports: [
  BrowserModule
],
providers: [],
bootstrap: [AppComponent]
})

export class AppModule { }
```

Un code qui demande à être un peu commenté :

Declarations

C'est un tableau des composants qui seront utilisés dans l'application.

```
declarations: [
  AppComponent,
  MonNouveauComponent
]
```

Import

C'est un tableau des différents modules de l'application. C'est dans cette section que l'on déclare le module regroupant l'ensemble des composants.

Providers

C'est ici que seront déclarés tous les services utilisés dans l'application.

Bootstrap

On déclare ici le composant principal de l'application

Persistance de données

Maintenant que notre application a bien évolué, et que l'on peut même passé de la page de Login à la page d'accueil, il serait peut être intéressant de sauvegarder quelques données notamment notre session utilisateur.

Ionic propose un plugin de stockage de données facile à prendre en main. Le stockage se fait sous forme de paires clé / valeur et utilise une variété de moteurs de stockage différent, en choisissant le meilleur disponible en fonction de la plate-forme.

Lorsqu'il est exécuté dans un contexte d'application native, Ionic Storage vérifie d'abord la présence d'un plugin SQLite, beaucoup plus intéressant quand il s'agit de stocker des données d'une application mobile.

Si SQLite n'est pas disponible, c'est le cas par exemple lorsque l'on utilise notre application depuis un navigateur, Ionic Storage utilisera dans l'ordre IndexedDB, WebSQL et localstorage :

SQLite ==> IndexedDB ==> WebSQL ==> localstorage

Installation

Nous allons tout d'abord installer le plugin SQLite :

```
$ ionic cordova plugin add cordova-sqlite-storage

Adding cordova-sqlite-storage to package.json

Saved plugin info for "cordova-sqlite-storage" to config.xml
```

Ensuite, il nous suffit d'installer Ionic Storage

```
$ npm install --save @ionic/storage
```

On déclare ce nouvel élément dans le module Root, pour qu'il puisse être appelé depuis n'importe quelle page :

src/app/app.module.ts

```
//Modules
// ...
import { IonicStorageModule } from '@ionic/storage';

// ...
imports: [
  BrowserModule,
  HttpClientModule,
  ComponentsModule,
  DirectivesModule,
  IonicModule.forRoot(DuckCoinApp, {
    // tabsPlacement: 'top',
    backButtonText: 'Retour'
  }),
  IonicStorageModule.forRoot({
    name: '__duckcoindb',
    driverOrder: ['indexeddb', 'sqlite', 'websql']
  })
],
```

Utilisation

Il vous suffit simplement d'injecter Ionic Storage dans la classe où vous souhaitez l'utiliser :

```
import { Storage } from '@ionic/storage';

export class LoginPage {
  constructor(private storage: Storage) { }

  ...

  // On sauvegarde le statut de connexion
  storage.set('isConnected', true);

  // On peut aussi récupérer cette valeur pour s'assurer que l'utilisateur a le droit d'afficher une page
  storage.get('isConnected').then((val) => {
    console.log('Suis-je connecté ?', val);
  });
}
```

Plugins natifs

En plus de composants purement visuel qui vous permettent de mettre en forme votre application, Ionic propose également des plugins natifs cette fois-ci pour interagir avec les fonctions dites natives de votre téléphone mobile. Ces plugins pour la plupart basés sur Cordova, sont disponibles en assez grand nombre à l'adresse :

<https://ionicframework.com/docs/native/>

Avant d'installer un plugin, il faut s'assurer que le package Ionic native est bien disponible, ce qui devrait être le cas dans une installation Ionic classique. Mais au besoin, il suffit de faire :

```
$ npm install @ionic-native/core --save
```

Exercez-vous

1) Créez un nouveau composant nommé "transaction" qui affichera la liste des dernières transactions de la blockchain. Cette liste est à récupérer à l'adresse : <https://duckcoin.charlesen.fr/transactions>.

2) Appelez ce nouveau composant dans l'onglet Portefeuille

3) Créez une directive que vous nommerez "**bigger**". Celle-ci permettra d'augmenter la taille (font-size) de l'élément qui l'invoquerait.

4) Ajustons un peu notre page de login en enregistrant en base de données le login au clic sur le bouton de validation, et en l'affichant en page d'accueil de façon à ce que l'on ait ce message :

```
Bonjour leLogin,  
Duckcoin, c'est la monnaie qui rend aimable, redonne le sourire et change la face du monde.  
Elle a été faite pour des gens sympas, juste comme toi.
```

P.S. : vous aurez peut être sûrement de ça ;-) : <https://ionicframework.com/docs/developer-resources/forms/>

5) Nous avons introduit les composants natifs proposé par Ionic <https://ionicframework.com/docs/native/>. Utilisez un maximum d'entre eux et ajoutez-les à votre projet.

Chap 9 - Tests et débogages avancés

Une fois développée, votre application mobile ne sera pérenne dans le temps que grâce à des tests poussés et une bonne gestion des bugs. Il se peut qu'un certain nombre de petites anomalies soit présentes à votre insu après que vous ayez de développer, mais ce n'est pas bien grave si le gros des soucis est corrigé en amont : vos utilisateurs cibles vous diront merci, et surtout, vous ne décevrez pas.

Tester son application mobile

Depuis votre navigateur

C'est la manière la plus simple pour tester votre application et comme nous l'avons il suffit de saisir la commande "serve" pour lancer votre application mobile dans un navigateur :

```
$ ionic start -lc
```

Pour information :

- **l** : ce paramètre permet d'afficher votre application dans le mode lab, avec la possibilité de switcher entre les vues Android, iOS ou d'avoir les deux en même temps
- **c** : permet d'afficher les logs depuis la console (terminal)

Depuis Ionic View

Ionic View est une application mobile qui vous permettra de visualiser vos projets Ionic après les avoir "pushé" sur le cloud à partir de votre compte Ionic PRO. Nous y reviendrons plus en détails au [chapitre 10](#).

L'application Ionic View n'existe actuellement que pour Android et iOS¹. Pour commencer à utiliser Ionic View, il vous suffit de le télécharger sur un des stores :

- Google Play Store : <https://play.google.com/store/apps/details?id=com.ionicframework.view>
- Apple Store : <https://itunes.apple.com/us/app/ionic-view-test-share-ionic-apps/id1271789931>

Depuis l'emulateur

```
$ ionic cordova run android
```

Débogages

C'est sûrement la partie la moins drôle quand on développe en général, mais fort heureusement avec Ionic et Angular les choses sont un peu moins douloureuse, la plupart des bugs étant explicites et dans certains cas on indique même ce qu'il faut faire.

Faisons le tour de quelques méthodes qui vont vous aider à debugger efficacement votre application mobile.

L'indétrorable `console.log()` et `console.info()`

Vous avez sûrement du l'utiliser au moins une fois dans vos développements JavaScript.

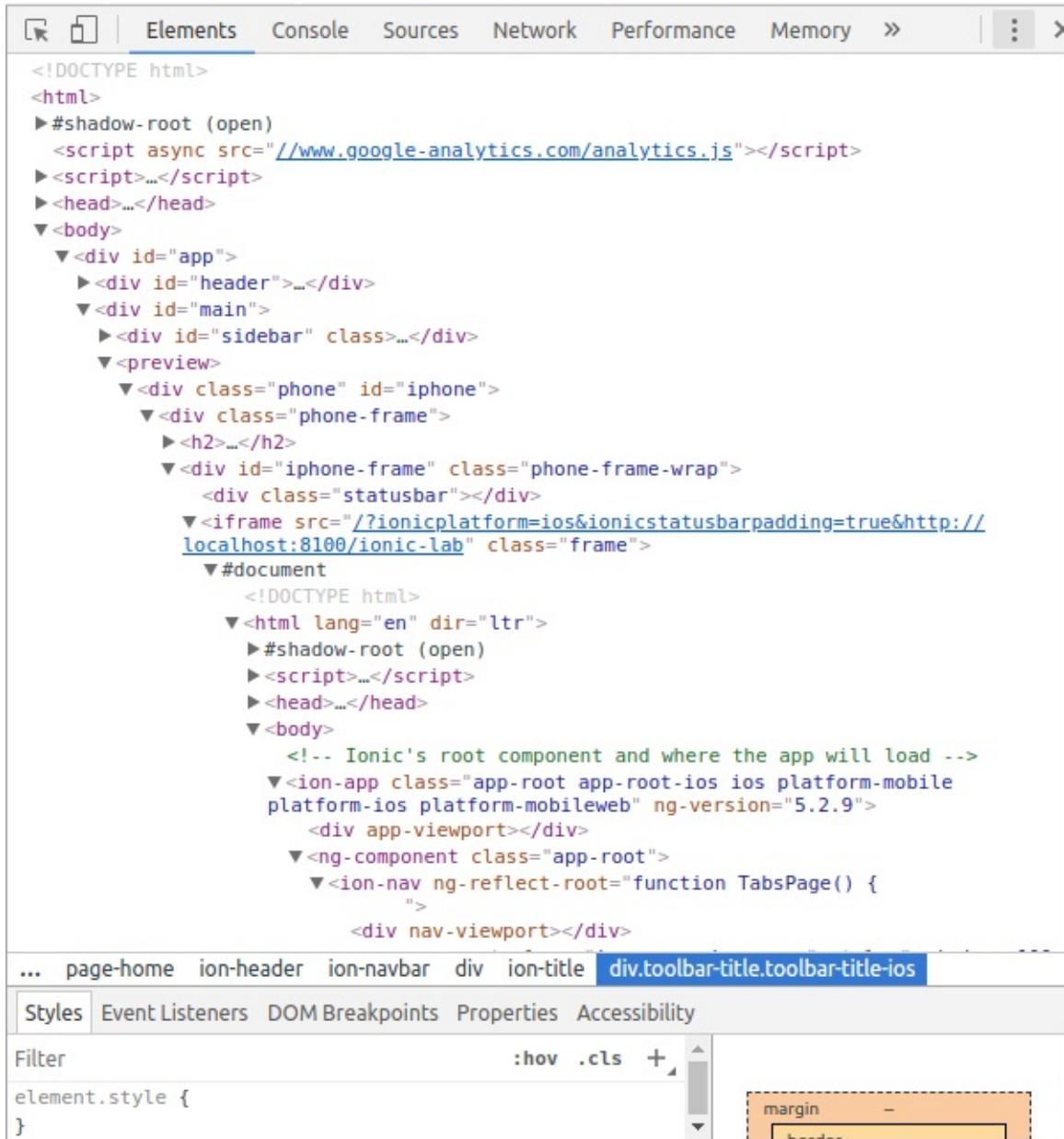
La méthode de logging de l'objet Console vous permet d'afficher le contenu d'une variable, du texte, un message...bref, à peu près tout, de votre code source vers le navigateur web.

```
console.log("Ceci est un message à mon navigateur web...Mayday ! Mayday !")
```

Il est possible de jouer sur le type d'affichage

Inspection du code source

La plupart des navigateurs dispose désormais d'un outil d'inspection de code source, disponible par exemple depuis la touche F12 ou clic-droit - **"Inspecter..."**.



Une application Ionic étant écrit en JavaScript, CSS et Html, vous n'aurez plus qu'à visualiser votre code et ajuster les choses au besoin : style CSS manquant ou non adapté, une image en 404,...

```
</ion-navbar>
</ion-header>
▼<ion-content padding class="content content-ios statusbar-padding" style="margin-top: 64px; margin-bottom: 49px;"></ion-content>
▼<div class="scroll-content" style="margin-top: 64px; margin-bottom: 49px;">
...
... page-home ion-header ion-navbar div ion-title div.toolbar-title.toolbar-title-ios
```

Styles Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

```
element.style {
```

```
.toolbar-ios-duckcoin .toolbar-title-ios, main.css:23661
.toolbar-ios-duckcoin .bar-button-clear-ios, .toolbar-ios-duckcoin .bar-button-default-ios {
  color: □#ffff;
}

.toolbar-title-ios { main.css:21999
  text-align: center;
  font-size: 1.7rem;
  font-weight: 600;
  color: ■#0000;
  pointer-events: auto;
}

.toolbar-title { main.css:21838
  display: block;
  overflow: ▶ hidden;
  width: 100%;
  text-overflow: ellipsis;
  white-space: nowrap;
}

* { main.css:5040
  -webkit_box_sizing: border_box;
  box-sizing: border-box;
  -webkit_tap_highlight_color: □ transparent;
  -webkit-tap-highlight-color: □ transparent;
  □ webkit_touch_callout: none;
}

div { user agent stylesheet
  display: block;
}

Inherited from ion-title.title.title-ios
```

```
.toolbar-ios ion-title { main.css:22007
  left: 0;
```

margin -
border -
padding -
195 x 20
-
-
-

Filter Show all

- box-sizing border...
- color □rgb(...
- display block
- font-family -apple-...
- font-size 17px
- font-weight 600
- height 20px
- overflow-x hidden
- overflow-y hidden
- pointer-events auto
- text-align center
- text-overflow ellips...
- text-rendering optimi...
- text-size-adjust 100%
- user-select none
- white-space nowrap
- width 195px
- word-wrap break-...
- -webkit-box-dir... normal
- -webkit-font-sm... antial...
- -webkit-tap-hig... □rgba...

Point d'arrêt (Breakpoint)

Il est également possible d'ajouter des points d'arrêts dans votre code source. Très pratique quand on se demande pour un bout de code ne fonctionne pas et que l'on souhaite avancer pas à pas jusqu'à trouver la source du problème.

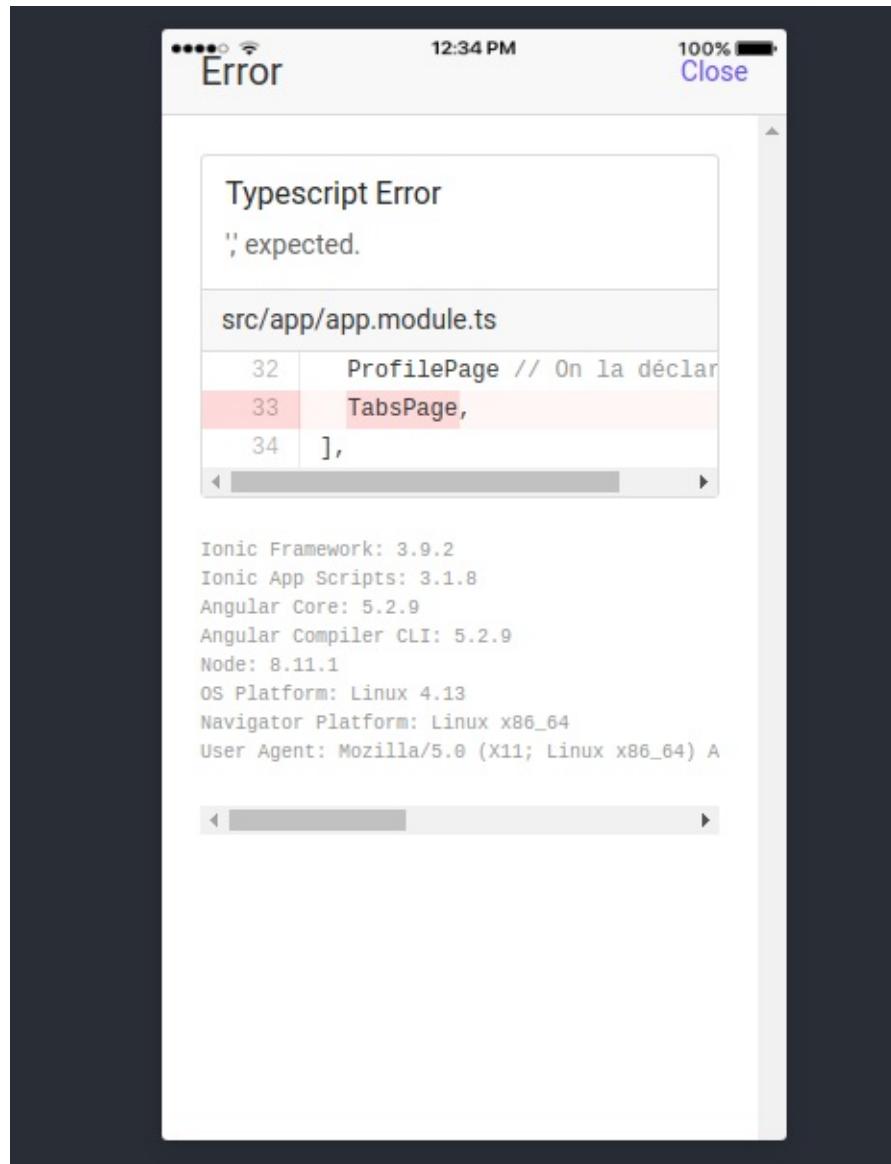
Vous avez le choix d'ajouter ce point d'arrêt depuis votre navigateur ou alors depuis votre code source à l'aide du mot clé "**debugger**" :

```
let maVariable = "Je suis une variable";
...
// L'exécution du code prendra fin ici...Pour avancer, faites F10 sur Chrome
debugger;
```

```
if (maVariable) {  
  alert(maVariable)  
}
```

Logs de la console + Gestionnaire d'erreurs Ionic

Ces deux éléments, assez vous indique souvent d'où vient le problème et surtout comment le résoudre. Il est parfois nécessaire de redémarrer votre projet depuis la console pour la recompilation permettre un retour de bug plus explicite, voir, dans de rares cas, corriger le bug.



```

L8: import {DirectivesModule} from '../directives/directives.module';
L9: import {PipesModule} from '../pipes/pipes.module';
L10: import { IonicStorageModule } from '@ionic/storage';

06:37:51] tslint: src/app/app.module.ts, line: 9
All imports are unused.

L8: import {DirectivesModule} from '../directives/directives.module';
L9: import {PipesModule} from '../pipes/pipes.module';
L10: import { IonicStorageModule } from '@ionic/storage';

06:54:27] build started ...
06:54:27] deeplinks update started ...
06:54:27] deeplinks update finished in 47 ms
06:54:27] transpile update started ...
06:54:28] build finished in 254 ms

06:54:35] tslint: src/app/app.module.ts, line: 9
All imports are unused.

L8: import {DirectivesModule} from '../directives/directives.module';
L9: import {PipesModule} from '../pipes/pipes.module';
L10: import { IonicStorageModule } from '@ionic/storage';

06:54:35] tslint: src/app/app.module.ts, line: 9
All imports are unused.

L8: import {DirectivesModule} from '../directives/directives.module';
L9: import {PipesModule} from '../pipes/pipes.module';
L10: import { IonicStorageModule } from '@ionic/storage';

]

]

```

Exercez-vous

1) Traquez tous les méchants bugs de votre application en utilisant une ou plusieurs des méthodes citées précédemment.

¹. L'application Ionic View pour iOS est à l'heure où j'écris ces quelques lignes indisponible sur l'Apple Store, ayant été désactivée par la plateforme qui lui reproche le fait d'être une application permettant d'exécuter d'autres applications. Selon Apple, chaque application doit disposer de son propre contexte d'exécution et ce que fait Ionic View enfreindrait sa politique interne. Voir l'article du CEO et co-fondateur Max Lynch [Update on Ionic View for iOS : https://blog.ionicframework.com/update-on-ionic-view-for-ios/](https://blog.ionicframework.com/update-on-ionic-view-for-ios/) ↵

Chap 10 - Ionic et son écosystème : Cloud, Lab, View et Creator

Ionic Cloud (PRO)

Nous avons déjà parlé et avons même créé un compte au [chapitre 3](#).

Ionic Cloud va vous permettre de compiler votre projet sans devoir vous inquiéter de l'installation des packages nécessaires pour chaque plateforme cible (Android, iOS).

Pour pusher votre application sur le cloud, et si vous avez choisi à la création du projet de l'héberger sur Ionic Pro, il vous suffira de faire :

```
$ git add .
$ git commit -m "Mon Application V 0.0.1"
$ git push ionic master
```

Il vous sera peut être demandé de vous authentifier si ce n'est déjà fait.

Si par contre vous avez choisi d'héberger votre projet sur Github, vous devrez simplement pusher vers votre branche **origin** :

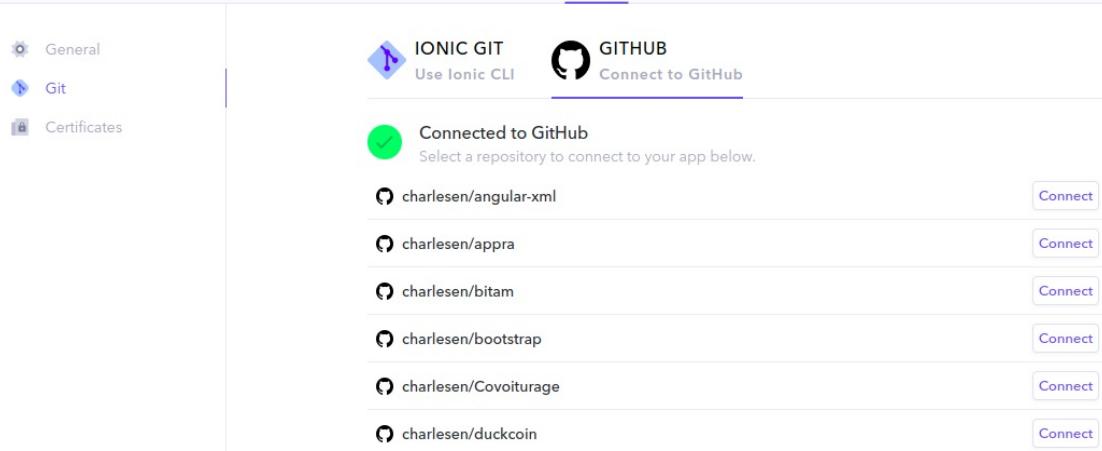
```
$ git add .
$ git commit -m "Mon Application V 0.0.1"
$ git push origin master
```

Si jamais vous souhaitez ne plus héberger votre application sur Github et tout rappatrier sur Ionic Cloud, il vous suffit de faire :

```
$ ionic link
? App ID 20e8461e is already set up with this app. Would you like to link it to a different app? Yes
```

Choisir "yes" à la question "**? App ID UNEAPPID is already set up with this app. Would you like to link it to a different app?**"

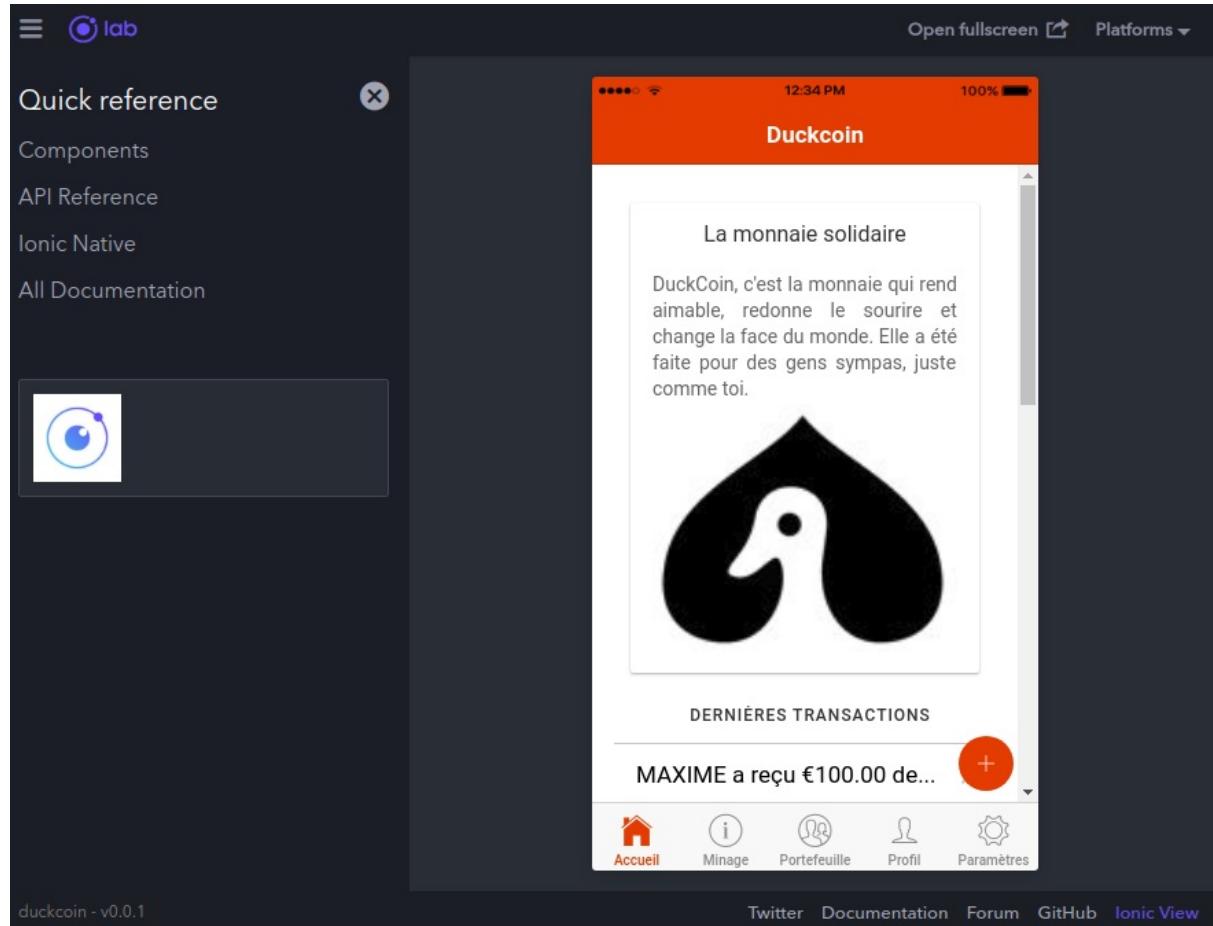
Choisissez ensuite Ionic Pro et l'outil s'occupera du reste. Puis dans votre espace Ionic, déconnectez votre application du service Git :



Url : <https://dashboard.ionicframework.com>

Ionic Lab

Il s'agit d'une vue spéciale de votre application sur un navigateur suite au lancement du projet depuis votre console. Vous avez la possibilité de tester le rendu de votre application pour iOS, Android et Windows Phone :

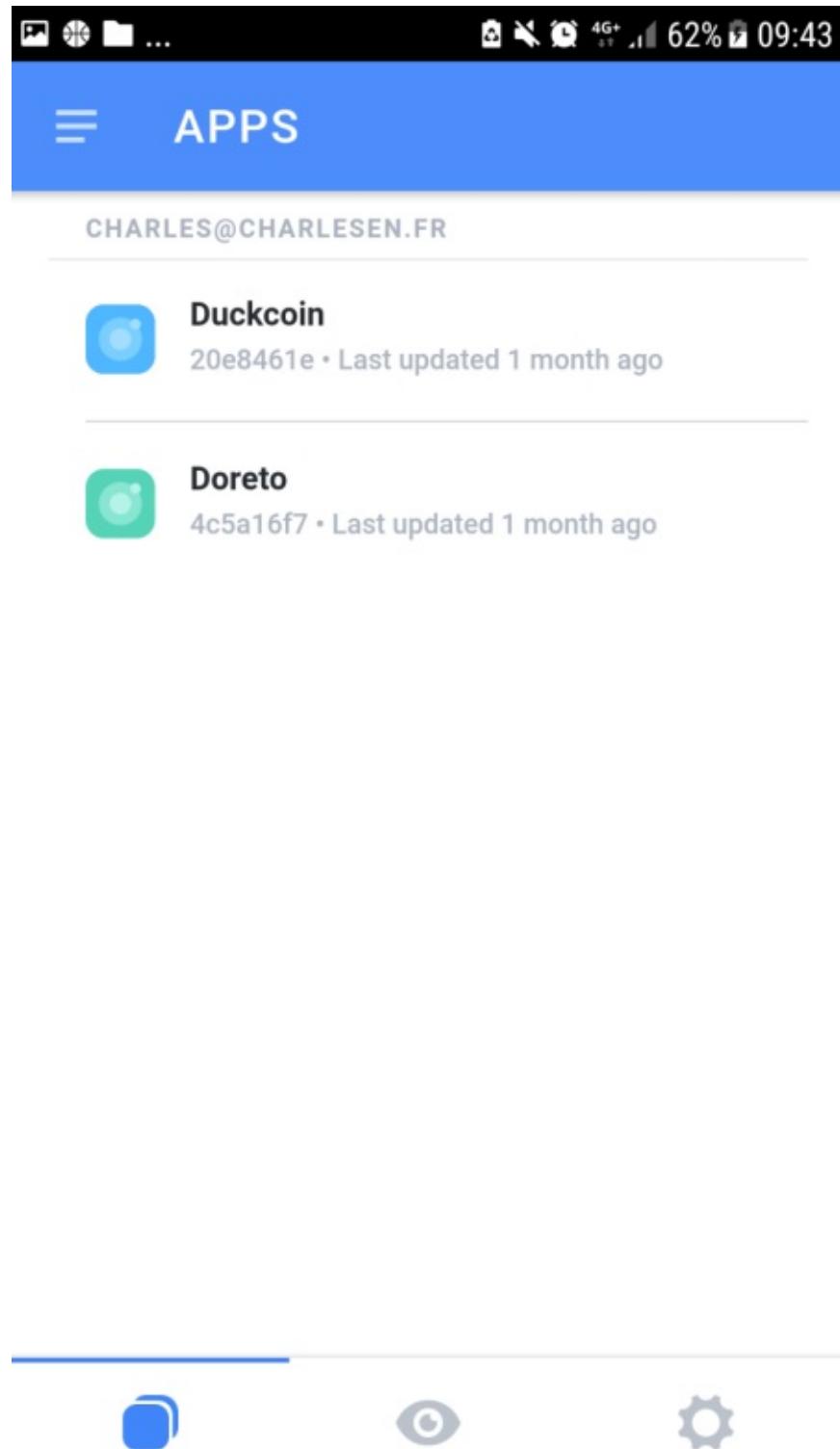


Ionic View

C'est l'application qui vous permettra de tester d'autres applications. On peut aussi la configurer depuis Ionic PRO.

Url : <https://dashboard.ionicframework.com>

Une fois l'application téléchargée, il vous suffit de l'ouvrir, de vous connecter avec vos identifiants Ionic PRO, pour voir s'afficher vos différentes applications :





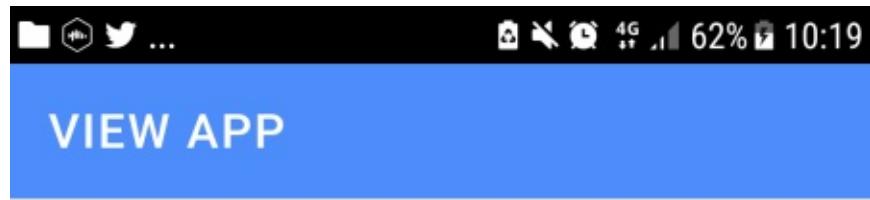
SHAKE DEVICE

Shake the device to access the Control Menu to send feedback on the app you're viewing or to exit the preview.

Loading...

Grâce au mécanisme de push avec Git, chaque mise à jour de code mettra aussi à jour votre application sur Ionic View.

L'onglet suivant vous permet de visualiser une application à partir de son identifiant ou de son QR Code :



Enter Code

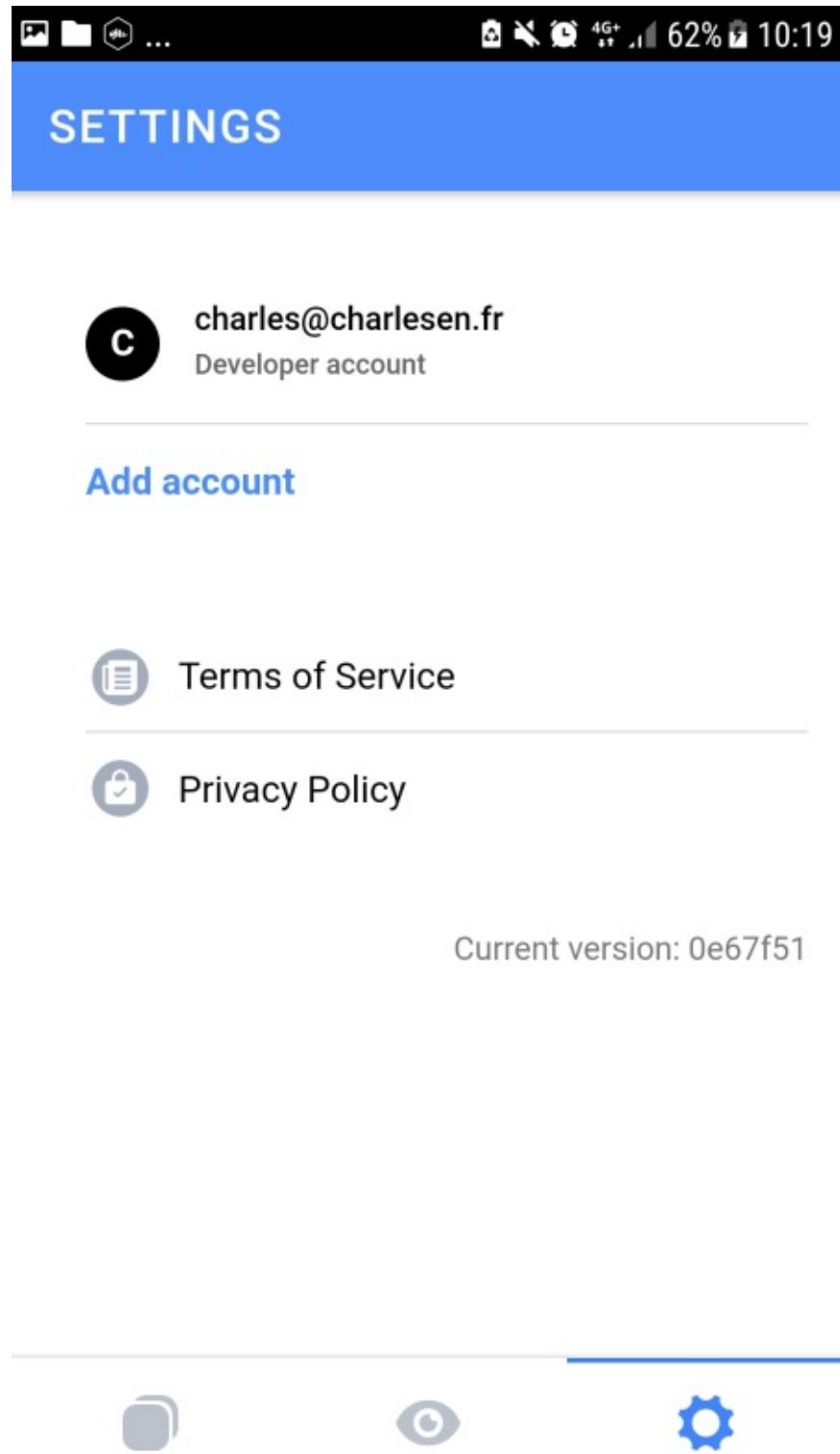
Enter a public AppID or Invite Code emailed to you.

ABCD1234

VIEW APP

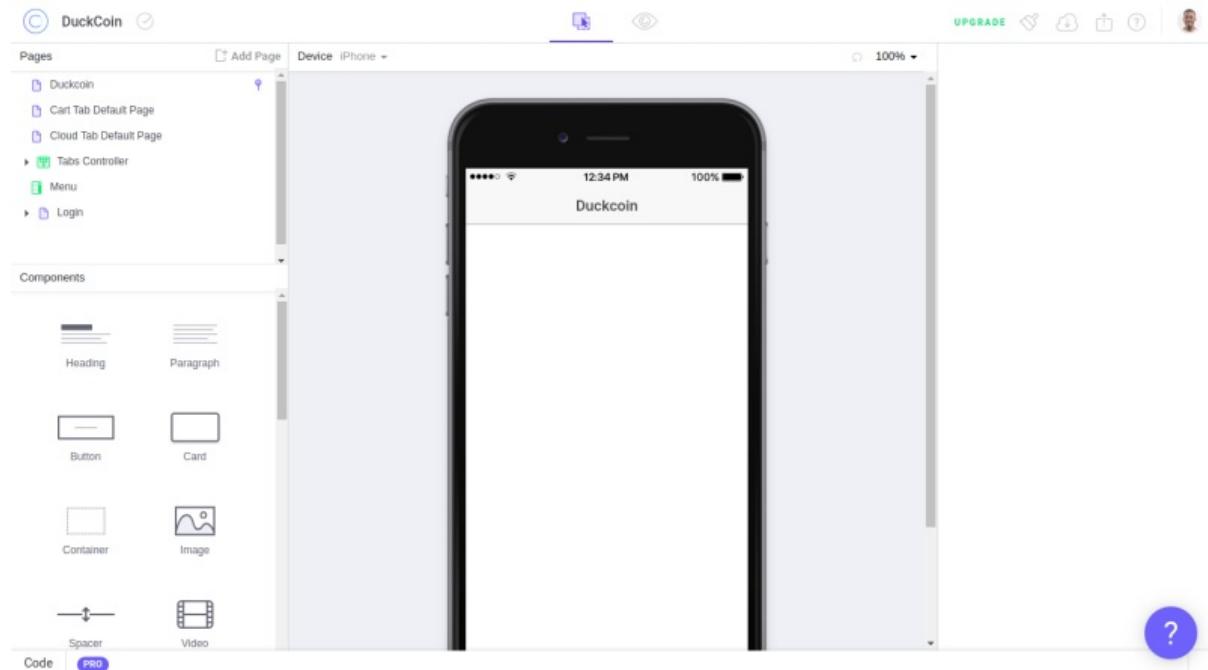


Et le suivant permet la gestion de ses paramètres utilisateurs



Ionic Creator

Creator est le logiciel *Wysiwyg* qui va vous permettre de créer votre application, sans saisir de code. Un éditeur en ligne vous est proposé, pour produire maquette, voir appli de production en quelques minutes.



Une fois votre travail effectué, il est possible d'exporter le projet.

A screenshot of the 'Export Your App' dialog box. The title is 'Export Your App'. Below it are four export options: 'ZIP FILE' (selected), 'APP STORE', 'PACKAGE', and 'CREATOR MOBILE APP'. A sub-section titled 'Download a .zip file of your project.' explains that the .zip file contains HTML/CSS/JS files in a standard organizational format. It also includes a 'Read the ZIP Tutorial' link and a large blue 'Download project ZIP' button. At the bottom right, it says 'Creator Project V2.0.0'.

Certaines fonctionnalités ne sont pas disponibles pour la version gratuite.

Url : <https://creator.ionic.io>

Ionic market

Le market place de Ionic propose des plugins, templates ou du tout en un prêt à l'emploi. Pratique quand on a peu ou pas d'inspiration, ou quand ne souhaite juste pas réinventer la roue.

Url : <https://market.ionicframework.com/>

Chap 11 - Publication sur les stores

Après le développement de votre application, la prochaine étape sera de la publier sur les stores et si ce n'est pas forcément l'étape la plus compliquée, elle peut s'avérer vite complexe, en particulier pour la publication sur l'Apple Store.

Publication sur le Google Play Store

Pré-requis

- [Java SDK](#)
- [Android Studio](#)

Création d'un Keystore

le keystore est utilisé pour signer une application Android et donc certifier qu'elle a bien été émise par nous. Nous allons utiliser un outil nommé **keytool** inclus dans le SDK de Java.

Depuis la racine de votre projet, lancez la commande suivante :

```
$ keytool -genkey -v -keystore duckcoin_app.keystore -alias duckcoin_app -keyalg RSA -keysize 2048 -validity 10000

Entrez le mot de passe du fichier de clés :
Ressaisissez le nouveau mot de passe :
Quels sont vos nom et prénom ?
[Unknown]: EDOU NZE Charles
Quel est le nom de votre unité organisationnelle ?
Quel est le nom de votre entreprise ?
Quel est le nom de votre ville de résidence ?
Quel est le nom de votre état ou province ?
Quel est le code pays à deux lettres pour cette unité ?
Est-ce CN=EDOU NZE Charles, OU=Mobile-tuts, O=Mobile-tuts, L=Troyes, ST=Champagne-Ardenne, C=FR ?
```

Retenez bien le mot de passe saisi, car une fois que votre application sera publiée sur le Google Play Store à l'aide de ce keystore, vous n'aurez pas d'autres choix que d'installer des mises à jour en vous basant sur celui-ci. Si vous perdez le fichier keystore, il vous suffira de le régénérer avec ce mot de passe. Si par contre vous perdez les deux et qu'une version de votre application existe déjà sur le store, vous ne pourrez plus la mettre à jour.

De plus, choisissez un alias assez à retenir. Le nom de votre application + '**_app**' devrait suffire. Par exemple **duckcoin_app**.

La commande a généré un fichier `duckcoin_app.keystore` à la racine du projet. Gardez le précieusement et à l'abri du vol, ce serait dommage qu'un parfait inconnu publie une version falsifiée de votre application et se faisant passer pour vous.

Création d'un package de production

Pour créer un package Android prêt à la production, il vous suffit de saisir la commande suivante :

```
$ ionic cordova build android --prod --release
```

Cette commande va générer un package **.apk** : `platforms/android/build/outputs/apk/monProjet.apk`.

Il faut ensuite signer ce package :

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore duckcoin_app.keystore monProjet.apk duckcoin_app
```

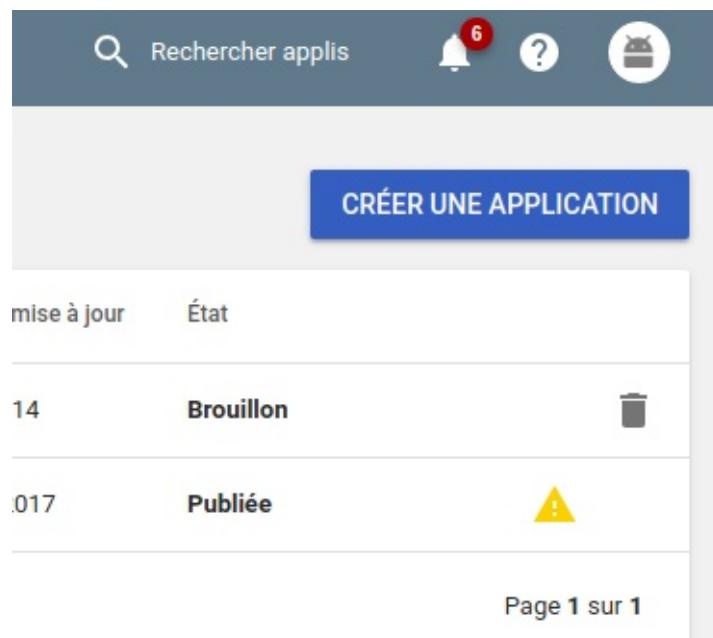
Une fois l'application signée, la dernière est la compression du paquet avec un outil inclus dans le SDK d'Android :

```
$ zipalign -v 4 monProjet.apk monProjetMini.apk
```

Voilà. C'est ce fichier **monProjetMini.apk** que nous allons pouvoir publier sur le store.

Google Play Store

Une licence délivrant le droit de publier sur le Google PlayStore coûte 25 dollars pour un compte à vie. Pour se connecter il suffit simplement d'aller à cette adresse : <https://play.google.com/apps/publish/>. Cliquez ensuite sur le bouton "Créer une application" en haut en à droite :



Créer une application

Langue par défaut *

Français – fr-FR

Titre *

DuckCoin

8/50

ANNULER CRÉER

Remplissez le formulaire qui vous est proposé :

Google Play Console

Fiche Play Store

DuckCoin Brouillon

Informations sur le produit

FRANÇAIS – FR-FR Gérer les traductions

Vous devez renseigner les champs marqués d'un * avant la publication.

Titre *
Français – fr-FR DuckCoin 8/50

Brève description *
Français – fr-FR La monnaie solidaire sur mobile 31/80

Description complète *
Français – fr-FR DuckCoin, c'est la monnaie qui rend aimable, redonne le sourire et change la face du monde. Elle a été faite pour des gens sympas, juste comme toi. ENREGISTRER LE BROUILLON

Ajoutez des captures d'écran dans les formats recommandés et validez :

Il faut ensuite aller dans l'onglet "**Versions de l'application**" et cliquez sur "**Gérer la production**" > "**Créer une version**". C'est ici que l'on va pouvoir uploader notre APK avant de le publier :

Publication sur l'Apple Store

Pré-requis

- Xcode 7 ou supérieur
- iOS 9
- Compte Developpeur Apple : <https://developer.apple.com/>.

Tout comme Android, Apple requiert un compte développeur pour pouvoir publier une application mobile.

Création d'un package de production

```
> ionic cordova build ios --prod
```

Création d'un "Provisioning profile" et publication sur l'Apple Store

La procédure est un peu moins évidente que sur Android et vous conseille la documentation Ionic très imagée :

<https://ionicframework.com/docs/intro/deploying/>

- Procédure de déploiement d'une application sur les Store : <https://ionicframework.com/docs/intro/deploying/>

Chap 12 - Introduction au PWA avec Stencil et Capacitor

Une Progressive Web App c'est quoi ?

C'est une application disponible à la fois sur le web et sur le mobile, et qui se comporte comme une application mobile native.

Une étude a montré que bon nombre d'utilisateurs, autour de 20%, se désintéressaient d'une application du fait de sa relative complexité d'installation sur mobile. De plus, de nombreux utilisateurs sont hésitants à l'idée d'installer une application qui prendrait à la longue beaucoup trop de place sur leur téléphone à cause du cache, des fichiers statiques,...

Le PWA vient en quelque sorte corriger ces problèmes et bien d'autres, rencontrés dans l'univers du mobile.

Ionic à l'assaut du PWA

La société éditrice du Framework mobile n'a pas attendu pour s'attaquer à ce qui pourrait s'apparaitre au futur du web et du mobile. Elle a créé en l'espace d'un an deux projets pour faciliter la création de PWA : Stencil et Capacitor.

Stencil

Site internet : <https://stenciljs.com/>



Docs Demos PWAs Resources GitHub ↗



The magical, reusable web component compiler

[GET STARTED](#) [LEARN MORE](#)

 Watch launch video



Les créateurs de ce projet le définissent comme "**un outil pour construire des composants web modernes et des progressive web apps (PWA)**".

Stencil veut tirer parti des principales nouvelles fonctionnalités disponibles dans les navigateurs telles que les "**Web Components**"¹, une nouvelle spécification du W3C² permettant la création de composants compatibles avec tous les frameworks. Ceux-ci fonctionnent aussi bien dans Angular, React, Ember, que Vue, il fonctionne aussi avec jQuery ou même sans framework du tout, car le cœur de Stencil est la création de simples éléments HTML.

Exemple de composant web, ici qui affichera le drapeau du pays précisé dans l'attribut "country".

```
<flag-icon country="fr"></flag-icon>
```

Installation et prise en main

Stencil utilise NodeJS et un certain nombre d'autres outils déjà présent si vous avez correctement installé Ionic.

Le démarrage d'un nouveau projet peut se faire en clonant un template exemple disponible sur github.

```
$ git clone https://github.com/ionic-team/stencil-starter.git mon_app_stencil

Clonage dans 'mon_app_stencil'...
remote: Counting objects: 306, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 306 (delta 14), reused 17 (delta 10), pack-reused 279
Réception d'objets: 100% (306/306), 329.35 KiB | 0 bytes/s, fait.
Résolution des deltas: 100% (153/153), fait.
Vérification de la connectivité... fait.

$ cd mon_app_stencil
$ git remote rm origin
$ npm install
```

Vous aurez besoin d'installer le plugin sass à l'intérieur du projet :

```
$ npm install @stencil/sass --save-dev
```

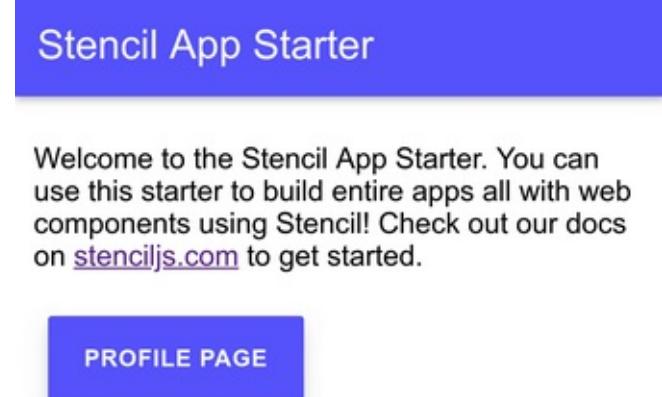
Et le déclarer dans le fichier de configuration :

```
const sass = require('@stencil/sass');

exports.config = {
  plugins: [
    sass()
  ],
  // ... suite du fichier de config
```

Il ne reste plus qu'à compiler tout cela, puis démarrer le projet :

```
$ npm start
```



Création de composants

Les composants créés avec Stencil sont simplement des classes écrites en TypeScript avec des décorateurs un peu spéciaux. La création d'un nouveau composant se fait à partir d'un fichier ayant l'extension .tsx.

Dans le dossier **src/components**, créons le fichier **moncomposant.tsx** avec le contenu suivant :

src/components/moncomposant.tsx

```
import { Component, Prop, State } from '@stencil/core';

@Component({
  tag: 'mon-composant',
  styleUrl: 'mon-composant.scss'
})
export class MonComposant {
  // On déclare ici les propriétés du composants
  @Prop() prenom: string;

  @Prop() nom: string;

  @State() isVisible: boolean = true;
```

```

    render() {
      return (
        <p>
          Salut, je m'appelle {this.prenom} {this.nom}
        </p>
      );
    }
  }

```

Créons aussi le fichier scss associé :

src/components/moncomposant.scss

```

mon-composant {
  background: #e43b00;
  padding: 5px;
  margin: 5px 0;
  border-radius: 5px;
  display: block;
  color: #fff;
}

```

Puis dans n'importe qu'elle fichier html, vous pourrez appeler ce composant comme n'importe quel autre tag html :

```
<mon-composant prenom="Charles" nom="EDOU NZE"></mon-composant>
```

Ajoutons par exemple ce tag en page d'accueil :

src/components/app-home/app-home.tsx

```

import { Component } from '@stencil/core';

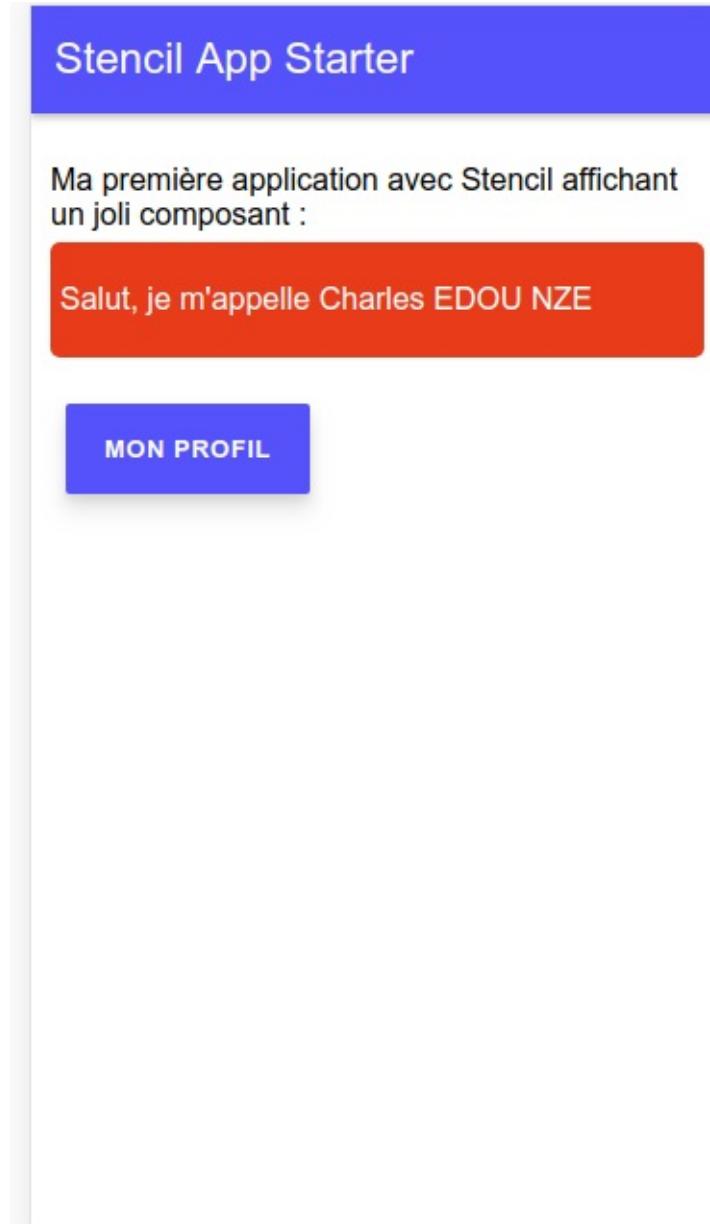
@Component({
  tag: 'app-home',
  styleUrl: 'app-home.css'
})
export class AppHome {

  render() {
    return (
      <div class='app-home'>
        <p>
          Ma première application avec Stencil
          <mon-composant prenom="Charles" nom="EDOU NZE"></mon-composant>
        </p>

        <stencil-route-link url='/profile/stencil'>
          <button>
            Mon profil
          </button>
        </stencil-route-link>
      </div>
    );
  }
}

```

ce qui donne ceci :



On peut aussi tout simplement rajouter ce composant au fichier index.html de l'application, après com :

```
$ npm run build
$ vi src/index.html # Choisissez votre éditeur préféré
```

```
<!DOCTYPE html>
<html dir="ltr" lang="en">
<head>
  <meta charset="utf-8">
  <title>Mon appli Stencil</title>
  <meta name="Description" content="Welcome to the Stencil App Starter. You can use this starter to build entire apps all with web components using Stencil!">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=5.0">
  <meta name="theme-color" content="#16161d">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta http-equiv="x-ua-compatible" content="IE=Edge"/>
```

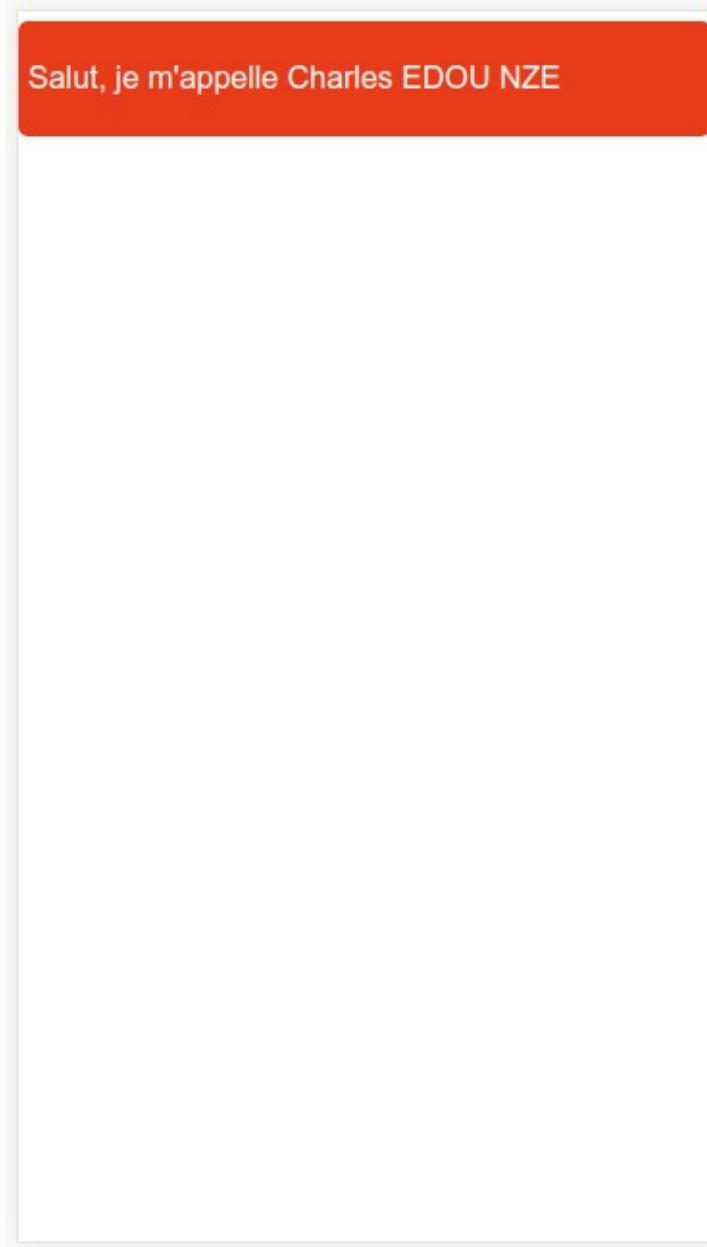
```
<script src="/build/app.js"></script>

<link rel="apple-touch-icon" href="/assets/icon/icon.png">
<link rel="icon" type="image/x-icon" href="/assets/icon/favicon.ico">
<link rel="manifest" href="/manifest.json">
</head>
<body>

<!--<my-app></my-app>-->
<mon-composant prenom="Charles" nom="EDOU NZE"></mon-composant>

<style>
  body {
    margin: 0px;
    padding: 0px;
    font-family: sans-serif;
  }
</style>
</body>
</html>
```

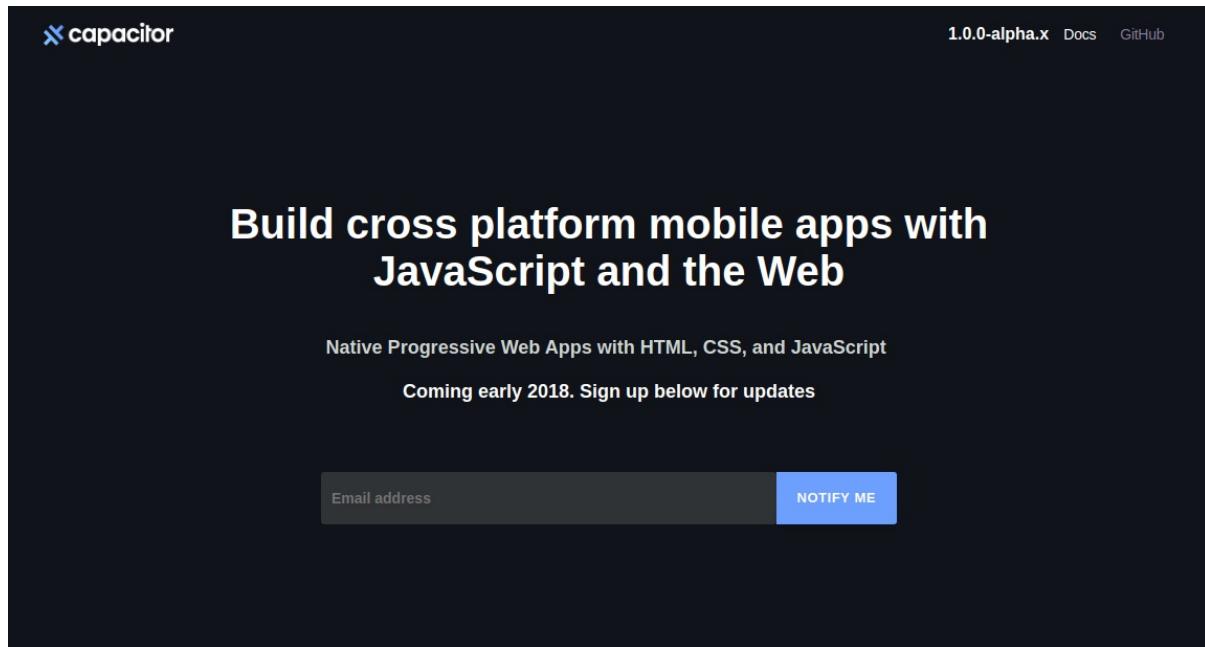
On n'affichera plus que le composant :



Salut, je m'appelle Charles EDOU NZE

L'idée est de pouvoir réutiliser des composants créés dans d'autres projets, les partager ou faire évoluer ceux des autres. C'est toute la philosophie des Web Components.

Capacitor : <https://capacitor.ionicframework.com/>



De ces deux outils, **Capacitor** est sûrement celui qui a le plus de potentiel et donc est susceptible de durer dans le temps. Mais les technologies mobiles évoluant tellement vite, seul l'avenir nous dira qui de ces deux technologies aura remporté le trophée de l'outil principal de développement de PWA.

Affaire à suivre donc.

¹. MDN web docs : https://developer.mozilla.org/fr/docs/Web/Web_Components ↵

². Ressources du W3C sur les WebComponents : <https://www.w3.org/wiki/WebComponents> ↵

Développement d'applications mobiles avec



Charles EDOU NZE

Ingénieur Etudes et Développement
spécialisé dans les technologies mobiles
et les systèmes embarqués, formateur et
fondateur du blog mobiletuto.com

www.mobiletuto.com

