PROGRAMMEZ!

le magazine des développeurs

#210 - septembre 2017

Angular 4

La version de maturité

Intelligence Artificielle

Utiliser les algorithmes génétiques en Java

Introduction au langage

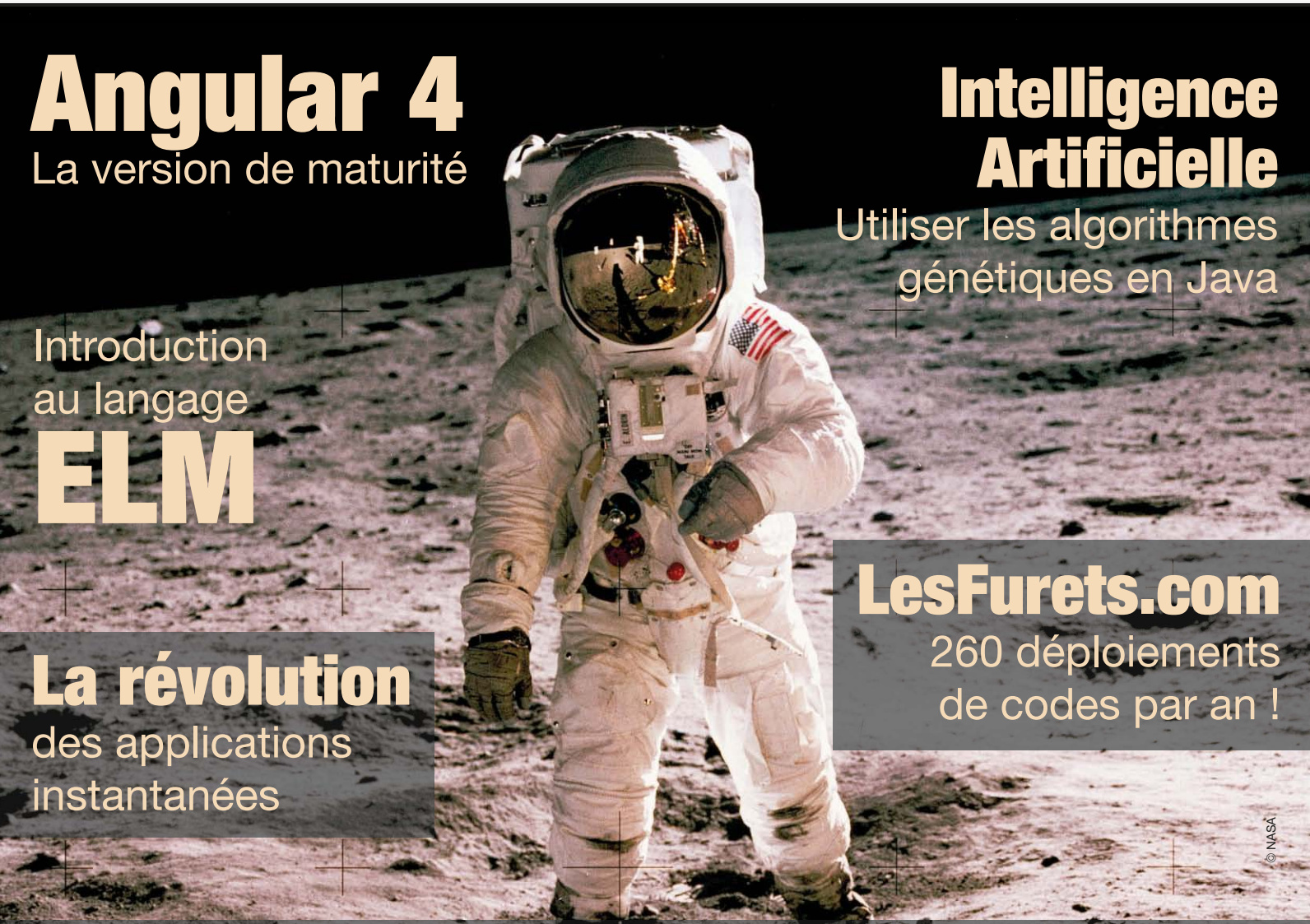
ELM

La révolution

des applications instantanées

LesFurets.com

260 déploiements de codes par an !



© NASA

Apollo 11

1969 : la NASA invente l'informatique et le métier de développeur

M 04319 - 210 - F: 6,50 € - RD

Le développeur full-stack est mort !

Devenez un développeur universel

Ma vie de développeur à... Hong Kong

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

Les héros ne meurent jamais

Il y a des technologies que l'on rêve de voir disparaître mais qui reviennent sans cesse, et d'autres qui disparaissent mais que l'on voudrait bien voir revenir. Adobe annonce la fin de Flash pour… 2020, certains doivent bien rigoler (comme un certain S.J.).
Les mauvaises langues vont dire "et Java c'est pour quand ?".

Parfois une technologie c'est tenace comme une mauvaise herbe. Rappelez-vous d'IE 6, ou même de Windows XP qui résiste à tous les désherbants, mais les derniers malwares vont avoir raison de lui…
D'autres sont de faux amis, n'est-ce pas Kernel Panic ou Blue Screen ? On vous aime bien (non, je rigole) mais franchement, pas la peine de venir trop souvent me voir.

Il y a aussi des prédictions que l'on peut reprendre tous les ans depuis 25 ans : "promis, juré, demain j'arrête Cobol et toute la partie mainframe". Mais non, chaque année, on se dit qu'ils sont toujours là. Et ils ne sont pas près de partir. On en prend au moins pour 20 ans, disons 30 pour être large. D'ici là, le cloud aura disparu, IE 6 aussi (soyons optimiste), Java en sera à la version 15 ou 19, selon la rapidité d'Oracle. Les drones livreront tout et n'importe quoi. Les robots domineront le monde et nous pourrons faire un "push to prod" un vendredi soir sans crise de panique.

Pour cette nouvelle saison - la 20e année ! - nous vous proposons un petit cocktail maison savamment dosé. Un peu d'Angular 4 ; vous aurez noté la mode de sauter un numéro de version. Ben oui, on passe de suite à une meilleure génération qui promet pas mal. Un peu de blockchain ; vous allez en manger de la blockchain cette saison ! On terminera (pour le moment) notre dossier Xamarin. On ajoutera un gros nuage d'Intelligence Artificielle, sans oublier un bon zeste bien velu du langage ELM (tu ne connais pas encore ? pas grave, on t'offre sa présentation, c'est cadeau ☺). Et on remue le tout avec de l'Apple 2 (les vieilles machines vaincront et le T-800, par ailleurs un robot super sympa, était dopé à l'Apple 2), un peu de Sharepoint Framework, d'Instant App, de Cobol.

Et si le cocktail de bienvenue du "bonheur est dans le code" vous a plu, on vous rajoute au menu de l'aventure spatiale avec la mission Apollo 11 (et comment la NASA inventa l'informatique moderne en 8 ans !). On parlera aussi de la vie d'un développeur qui vit et travaille à Hong Kong, avec tout plein de bons conseils pour survivre.

On va aussi t'aider, ami(e) développeuse / eur à être bien dans ta société, et pour que le grand patron, ou ton directeur technique, te dise "je bosse sur KDE. Tu me conseilles de passer à Gnome ou carrément au sous-système Linux de Windows 10 ?".

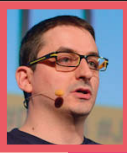
Et on t'expliquera comment faire 260 déploiements de codes par an…

Bonne rentrée.

François Tonic
ftonic@programmez.com

Agenda 6
Tableau de bord 4

Un développeur à Hong Kong 10

Chronique sur les tests 12
ABONNEZ-VOUS ! 11
Des développeurs épanouis en entreprise Partie 1 16



Blockchain 22

Pi + serveur web 19

Réalité virtuelle 27
Angular 4 28


Retour terrain 36

Cobol 40




Java & IA 50
Xamarin Partie 2 43
API hypermedia Partie 2 54


Langage ELM 58
Sharepoint framework 63
Apollo 11 70
Instant apps 67
Pascal & Apple II Partie 2 79
Développeur universel 75


Dans le prochain numéro !
Programmez! #211, dès le 30 septembre 2017

Choisir sa base de données
Quelle base de données choisir ? Quels critères techniques retenir ? Version locale ou version cloud ? Les réponses dans notre dossier spécial.

Linux au coeur de Windows 10
Le sous-système Linux de Windows 10 offre des usages inédits pour les développeurs et les sysadmins. Présentation complète.

C++ : le langage de référence !
Mature, robuste, performant, multiplateformes, il a tout pour lui. Découvrez pourquoi C++ est toujours un langage de référence en 2017 !

HyperCard d'Apple a fêté son 30e anniversaire. Il fut distribué avec les Macs à l'été 1987 avant d'être définitivement arrêté en 2004 après une dernière mise à jour en 1998. Que de souvenirs !

Red Hat a décidé de bannir le système de fichiers Btrfs que l'on retrouve dans les NAS de Synology. Ce système est réputé par sa fiabilité. Pour son système Linux, l'éditeur mise sur ZFS. Tous les chemins mènent à Oracle…

Facebook lance son site "marketplace" en Europe, le propriétaire du site LeBonCoin, Schibsted, a vu son action en bourse chuter. Il faudra attendre quelques mois pour mesurer l'impact de l'arrivée d'un concurrent de poids.

Intel, sous pression des processeurs ARM, a présenté mi-août une nouvelle architecture processeur : Ice Lake. Jusqu'à présent, Cannon Lake succédait à Coffee Lake qui lui-même succédait à Kaby Lake. Cannon Lake doit arriver en 2018 en gravure 10 nm. Intel n'a pas donné beaucoup de détails sur Ice Lake qui se présenterait comme le successeur de Cannon Lake (ou est-ce Coffee Lake ?). Mais en toute logique, Ice Lake serait d'abord pour les puces dédiées aux ordinateurs portables avant d'être généralisée.

CULTUREGEEK

Pas mal de bonnes choses à regarder :

- Halt and Catch Fire : la 4e et dernière saison a démarré le 19 août dernier. Le dernier épisode de la S3 nous avait laissés sur une image forte : la naissance de l'Internet et d'un ordinateur NeXT. Dix épisodes à savourer. Autant la 1ere saison était géniale, autant les S2 et S3 manquaient parfois de rythme avec des épisodes assez inégaux.
- Mr Robot : la saison 3 démarrera le 11 octobre. La fin de la S2 était top. Impatient !
- Star Trek Discovery : à partir du 25 septembre sur Netflix. La saison 1 sera diffusée en deux parties et comportera 15 épisodes.

Et aussi :

- Stargate Origins : mini-série de 10 épisodes de 10 minutes qui racontera la vie de Catherine Langford, la petite fille de l'explorateur qui découvre la porte des étoiles en 1928. Les épisodes seront disponibles cet automne sur le site officiel : stargatecommand.co
- Twin Peaks : la saison 3 a été l'un des évènements de cet été. Réussie ou râtée, la série fascine toujours. Le dernier épisode se dévoilera le 3 septembre aux Etats-Unis. Pas de saison 4 pour le moment.
- X-Files : la saison 11 est toujours prévue pour 2018, avec 10 épisodes.
- The man in the high castle : la superbe série d'Amazon reviendra pour une 3e saison. Diffusion fin 2017 ?

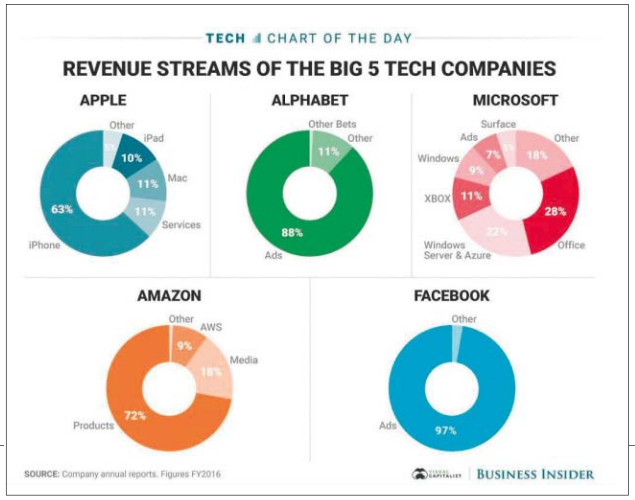
INDEX TIOBE

Comme chaque mois, l'index TIOBE montre la popularité des recherches par langage de programmation. Pour août, pas de changement pour les 5 premiers langages : Java, C, C++, C# et Python. La suite connaît quelques évolutions : VB.net connaît une remontée ainsi que Ruby. PHP et JavaScript reculent légèrement. Swift est désormais à la porte du top 10; le langage est actuellement 11e. A noter les progressions fortes de Go et de Dart. Kotlin est encore très loin : 41e…

08/17	08/17	Evolution classement	Langage	% des recherches	Evolution
1	1	=	Java	12.961%	-6.05%
2	2	=	C	6.477%	-4.83%
3	3	=	C++	5.550%	-0.25%
4	4	=	C#	4.195%	-0.71%
5	5	=	Python	3.692%	-0.71%
6	8	+2	Visual Basic .NET	2.569%	+0.05%
7	6	-1	PHP	2.293%	-0.88%
8	7	-1	JavaScript	2.098%	-0.61%
9	9	=	Perl	1.995%	-0.52%
10	12	+2	Ruby	1.965%	-0.31%

LES DÉPENDANCES DES GAFA(M)

On parle souvent des méchants GAFA(M) (Google, Amazon, Facebook, Apple, Microsoft). Mais il est tout aussi intéressant de voir sur quoi cette puissance économique repose. Business Insider nous rappelle des faits simples : certaines sociétés dépendent d'un seul produit. Bref, elles font de la "monoculture", un peu comme en agriculture quand un exploitant dépend d'une unique culture. C'est une force et une faiblesse. Une force quand tout va bien, que le marché est porteur, une faiblesse quand le marché stagne et baisse. Trois de nos GAFA(M) sont hyperdépendants : Alphabet (dont fait partie Google), Amazon et Facebook. Alphabet et Facebook dépendent énormement des revenus publicitaires, c'est particulièrement criant avec Facebook. Amazon, grâce à son site marchant ne repose pas que sur les services cloud et les medias. Apple est certes moins dépendant mais l'iPhone écrase les autres produits et services (63 %). Ce qui pose la question à court terme (3-5 ans) du produit qui va pouvoir prendre le relais de l'iPhone.



Le seul qui arrive à diversifier réellement ses revenus est Microsoft. On constate tout de même deux gros revenus : Office et Windows Server – Azure. Windows proprement dit représente à peine 9 %, l'éditeur a donc su muter et réduire sa dépendance à Windows. Mais cette répartition est aussi une relative faiblesse car l'éditeur n'a pas un réel leader qui se démarque. Mais cette répartition est plus sécurisante sur le long terme.

WINDEV®

DÉVELOPPEZ 10 FOIS PLUS VITE

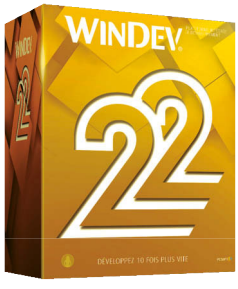
WINDEV 22: ATELIER DE DÉVELOPPEMENT PROFESSIONNEL, COMPLET EN STANDARD

Gestion du cycle de vie complet: Idée, Conception, Développement, Génération, Déploiement, Exploitation • Un code multi-plateformes Windows, Linux, Java, Internet, Mobiles • Environnement ALM complet • Toutes les bases de données sont supportées, Big Data • Inclus: HFSQL, base de données locale, Client/Serveur, cluster, embarquée et cloud • Puissant RAD • Intégration continue • Tableau de bord de vos applications • Audit statique & dynamique • Générateur de fenêtres (UI) visuel & intuitif • Utilisation facile de charte graphique • Héritage et surcharge d'interface • Tous les champs (contrôles) sont très puissants et livrés en standard: Champ de saisie, Tableau croisé dynamique (cube), Champ Planning, Champ Diagramme de Gantt, Champ Tableau de bord, Champ Table, Champ Graphe, etc • FAA: chaque application bénéficie automatiquement de Fonctionnalités Automatiques: export vers Excel, vers Word, envoi d'email, etc • Sécurité : Mot de passe de vos applications • Puissant générateur de rapports et codes-barres • Langage de 5ème génération: WLangage • Editeur de code intuitif avec puissant débogueur • Tests unitaires et tests automatiques • Versioning (GDS/SCM) • Webservices SOAP et Rest • Modélisation Merise et UML • .NET, 3-Tier, MVP • Support de tous les standards: XML, USB, Bluetooth, NFC, J2EE, OLE, ActiveX, RPC, SaaS, SMTP, FTP, OPC, DLNA, IoT, Sockets, API, Webservices... • Lien avec Lotus Notes, SAP, Google, Outlook • Multimédia, Domotique • Livré avec des centaines d'exemples et d'assistants • Génère le Dossier technique d'un clic • Télémétrie pour connaitre l'utilisation réelle de vos applications • Générateur d'aide • Support de 64 langues étrangères par application • Générateur de procédures d'installation: local, CD, USB, Internet, Réseau, Push... • Robot de surveillance: surveillez vos applications • Gestion des suggestions et incidents utilisateurs • Support Technique Personnalisé Gratuit* • ...

CONSULTEZ PLUS DE 100 TÉMOIGNAGES SUR LE SITE PCSOFT.FR

Elu
«Langage le plus productif du marché»

VERSION EXPRESS **GRATUITE**
Téléchargez-la !



Logiciel professionnel. * 15 requêtes gratuites sur la version en cours de commercialisation

Tél 04 67 032 032

WWW.PCSOFT.FR

septembre

UBUCON EUROPE

8, 9 &10 septembre / Paris

La communauté Ubuntu se réunira à Paris début septembre à l'UbuCon. Il s'agit de la 2e édition de la conférence européenne, la 1ere édition s'était déroulée en Allemagne. L'évènement s'annonce important : des dizaines de conférences et d'ateliers, 16 salles, 2000 m2 dédiés. Les communautés seront naturellement très présentes avec des espaces pros, communautaires et de nombreux thèmes seront abordés : système, smart city, infrastructure, etc.

Pour en savoir plus : https://ubucon.paris

SQLSATURDAY

16 septembre / Toulouse

La communauté SQL Server se réunit pour une grande conférence internationale pour les développeurs et DBA. Il s'agit de la première édition.

Pour en savoir plus : http://www.sqlsaturday.com

DEVFEST TOULOUSE 2017

28 septembre / Toulouse

Le DevFest est une conférence technique destinée aux développeur.se.s. Elle s'adresse aussi bien aux étudiants, aux professionnels ou tout simplement aux curieux technophiles. Durant toute la journée, des orateurs & oratrices reconnus viendront présenter des sujets variés : autour du développement mobile, du web, de la data, des objets connectés, du cloud, du devops, etc... ainsi que des bonnes pratiques de développement. Le DevFest Toulouse est organisé par les communautés de développeurs de Toulouse, et porté administrativement par le GDG Toulouse.

Plus d'infos sur : https://devfesttoulouse.fr

RGC 2017

30 septembre & 1er octobre / Meaux

Pour tous les amoureux de jeux et d'ordinateurs anciens, la RGC est une convention qui réunit plus de 300 personnes en automne. L'édition 2017 promet d'être très riche et active sur l'ensemble des machines : Atari, Amiga, MSX, CPC, etc. La RGC est réservée aux inscrits.

octobre

MICROSOFT EXPERIENCES'17

4 & 5 octobre / Paris

Cette année, les MS Experiences parleront de 3 grands thèmes : intelligence artificielle, les nouvelles méthodes et pratiques de travail, et la confiance numérique (blockchain, identité, etc.). L'événement parlera bien entendu aux responsables IT et aux développeurs. Le format reste grosso modo identique à l'édition 2016 : plénières, ateliers, sessions.

FORUM PHP 2017

26 & 27 octobre / Paris

L'édition 2017 se transporte de nouveau à Paris, à 2 pas de Denfert-Rochereau. 190 propositions de sessions ont été déposées. Un des focus sera le langage proprement dit : PHP 7 et l'avenir du langage.

Pour en savoir plus : event.afup.org

novembre

NI DAYS PARIS

7 novembre / Paris

National Instruments est un constructeur / éditeur phare du monde de l'embarqué et de l'industrie. Chaque année, la journée NI Days permet de découvrir les dernières versions des outils d'instrumentation et l'écosystème. De nombreuses sessions sont jouées durant la journée. Pour en savoir plus : https://www.ni.com/nidays/

OPENSTACK DAY FRANCE

21 novembre / Paris

OpenStack Day France sera l'opportunité de rencontrer des représentants de la Fondation OpenStack, les acteurs de l'écosystème du cloud en Open Source, les utilisateurs finaux, et les contributeurs de la communauté de développeurs en France.

Pour en savoir plus : openstackdayfrance.fr

LEAN KANBAN FRANCE 2017

29 & 30 novembre / Paris

5e édition de la conférence dédiée à Kanban. Des sessions en Anglais et en Français pour débutants et experts, venez découvrir : des retours d'expérience (Blablacar, LesFurets, ...), Niels Pflaeging et Dan Mezick sur le leadership et les entreprise, Joanne Molesky co-auteure de Lean Enterprise, Patrick Steyaert sur le kanban upstream et David Anderson initiateur du mouvement Lean Kanban sera présent pour nous partager les dernières évolution de cet univers. Des masterclass sont prévues avec certains orateurs, merci de consulter le site pour plus d'informations.

Site http://www.leankanban.fr

Contact: contact@leankanban.fr

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT

*CLOUD**IKOULA**ONE* est une solution de Cloud public, privé et hybride qui vous permet de déployer **en 1 clic et en moins de 30 secondes** des machines virtuelles à travers le monde sur des infrastructures SSD haute performance.

www.ikoula.com sales@ikoula.com 01 84 01 02 50

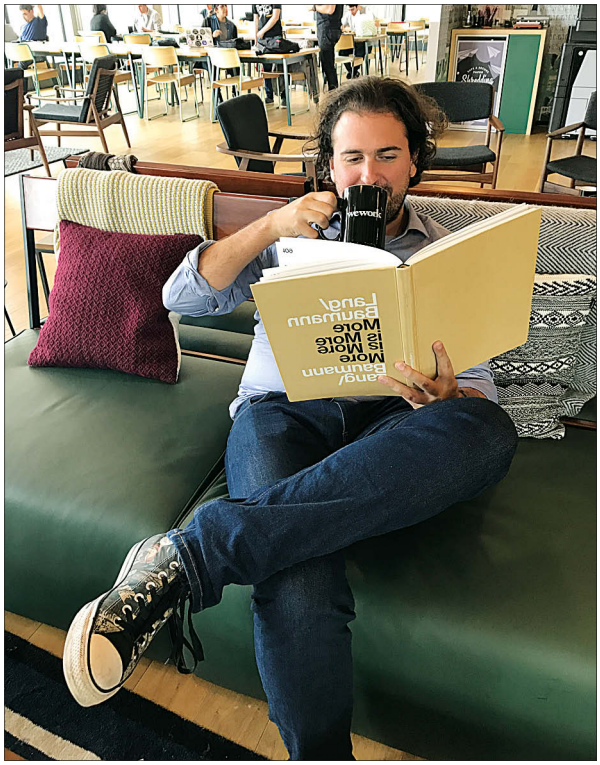
ikoula HÉBERGEUR CLOUD

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL

Clément Bitton, expert données de PALO IT à **Hong Kong**

En hommage à Clément

Clément Bitton est aujourd'hui expert données chez PALO IT Hong Kong. Il évoque pour Programmez ! son expérience en Asie et les petits conseils pour réussir son installation et travailler dans une culture qui n'est pas la nôtre.



Pourquoi as-tu choisi de quitter la France pour travailler en Asie ?

En 2007, j'ai effectué ma dernière année d'études à Taiwan dans le cadre d'un programme d'échange scolaire. Baignant dans une culture internationale depuis ma plus tendre enfance, et ayant eu l'occasion de travailler quelque temps aux Etats-Unis, je souhaitais poursuivre ma carrière a l'étranger. A la suite d'un différend avec l'un de mes employeurs, j'ai décidé de tout plaquer et de tenter ma chance ailleurs. Je m'étais donné six mois, dont trois mois à Hong Kong, puis trois mois à Singapour (en cas d'échec à Hong Kong). Pourquoi ces deux villes ? Tous simplement car ayant déjà été en Asie, je connaissais Hong Kong et je savais que cette ville était très festive. Côté Singapour, il y avait à l'époque plus d'offres d'emploi en BI & Big Data qu'ailleurs en Asie.
De plus, en Asie, le climat est très doux. Ça tombe bien : je n'aime pas le froid ! Je dois admettre que dans l'avion me menant à Hong Kong, je me suis répété plusieurs fois "Mais qu'est-ce que tu fais ? Tu vas te planter !". Puis, dans les semaines qui ont suivi, j'ai cherché, démarché, je me suis "incrusté" dans beaucoup d'événements pour networkers… tout en explorant la jungle urbaine qu'est Hong Kong ! Après deux mois, j'ai eu la chance de trouver un job chez Cathay Pacific, une compagnie aérienne asiatique (j'en profite d'ailleurs pour remercier Alexandre E. et Antoine V. qui ont fortement contribué à cette réussite !).

Parle-nous de ta première expérience professionnelle à Hong Kong

Chez Cathay Pacific, j'ai commencé comme développeur au sein de l'équipe BI pour la partie Engineering : suivi des incidents sur les avions, tracking des pièces détachées, etc. A force de pester sur les normes de développement, on m'a confié la mise en place des normes de leur nouvelle Data Warehouse. J'en profite pour préciser que la tendance naturelle des français à râler et critiquer, si c'est fait de manière constructive, est un atout considérable dans le monde du travail outre-hexagone !

A quelles différences culturelles et professionnelles as-tu été confronté ?

Il y a clairement d'énormes différences de culture professionnelle entre le monde français, le monde anglo-saxon et le monde Asiatique. Sans parler de la culture professionnelle d'autres nationalités d'expatriés venant travailler à Hong Kong. Constat : il faut apprendre à s'adapter à chacune. De mon point de vue, il faut distinguer trois points suivants dont j'ai volontairement forcé le trait :

La communication :

- Culture française : nous avons tendance à être très pointilleux sur les détails, à critiquer le message quel qu'il soit, pour montrer que nous en savons autant que l'autre. Si une erreur est faite, nous la mettons en évidence et demandons au responsable de la corriger.
- Culture anglo-saxonne : on recherche avant tout la valeur ajoutée du message. Même si l'on est critique. En cas d'erreur, l'accent est avant tout mis sur les réalisations positives, et il arrive de demander à une tierce personne de corriger l'erreur.
- Culture asiatique : on a tendance à ne pas dire les choses, voire même "tourner autour du pot". On évite de dire à une personne qu'elle a fait une erreur. Si une erreur a été faite, elle a été faite par une personne indéfinie, et elle sera corrigée si besoin.

Le management :

- Culture française et anglo-saxonne : le management est assez similaire. Généralement, le manager supporte ses équipes pour les aider à réussir. Le manager les assiste pour résoudre les problèmes et fait tampon en cas d'incident. Il suit les recommandations de ses équipes.
- Culture asiatique : les équipes supportent leur manager et font ce qu'il dit. Le manager est le chef et toutes les personnes sous ses ordres doivent le suivre. En cas de problème, ce sont aux équipes de traiter les points et de ne pas les rapporter au manager. Le manager décide des orientations à prendre et les équipes doivent s'exécuter, même si elles pensent que les choix sont mauvais.

La façon de travailler :

- Culture française : nous travaillons rapidement et efficacement. Peu importe le temps passé au travail. Nous sommes plutôt proactifs, avec l'envie d'en faire toujours plus. Nous n'hésitons pas à nous affranchir des procédures si besoin. Nous apprécions les normes de développement et les standards (en France, j'ai toujours travaillé avec de jeunes consultants passionnés, c'est donc de cette façon que je qualifierai la façon de travailler à la française).
- Culture anglo-saxonne : on travaille rapidement et efficacement, comme en France. Néanmoins, on ne reste pas plus tard que les horaires de bureau habituels. Si on reste plus longtemps, c'est

7 NOVEMBRE 2017 | PALAIS DES CONGRÈS DE PARIS

NIDays, c'est l'occasion de rejoindre les acteurs de l'innovation d'un grand nombre d'industries et de découvrir comment les progrès dans les technologies des transports, l'automatisation des tests, la communication 5G et autres encore se conjuguent pour créer un monde plus intelligent et connecté à travers des systèmes définis par logiciel.

INSCRIVEZ-VOUS DÈS MAINTENANT SUR NI.COM/NIDAYS

NATIONAL INSTRUMENTS™

*L'ingénierie du futur

signe qu'on est incapables de faire son travail dans les délais impartis. On est moins proactifs, mais avec l'envie de bien faire. On respecte autant que possible les procédures et on ne s'en affranchit pas sans autorisation. Les normes de développement ne sont pas nécessairement importantes. C'est plutôt la documentation et le code réutilisable qui le sont.

- Culture asiatique : on fait acte de présence et on reste au travail longtemps, même s'il n'y a rien à faire. La proactivité est quasi-nulle. On suit à la lettre les procédures, même si elles sont obsolètes. Il n'y a aucune norme de développement. Le code n'est pas réutilisable, et la documentation est nulle.

Avec le recul, je dirais que l'adaptation la plus complexe est celle de la communication.


Vue de Hong Kong

Y-a-t-il des différences importantes en matière de technologies et de pratiques ?

Hong Kong est très, très en retard quant au Big Data. Ou plus généralement en termes d'analytics. Je pense que cela est dû au fait que la mise en place de ces derniers implique une capacité de remise en cause des décisions de la part du management. Ce qui est en totale contradiction avec la culture de "ne pas perdre la face". Par conséquent, la mise en place des projets de BI ou de Big Data est souvent retardée.

Quelle est ta situation professionnelle actuelle ?

Après 3 ans chez Cathay, je commençais à m'ennuyer et voulais donc tenter autre chose. A la suite d'une aventure peu concluante dans une startup, j'ai choisi de rejoindre PALO IT Hong Kong, en suivant la recommandation d'une amie chasseuse de tête. J'étais intéressé par le monde du consulting car il permet de couvrir différentes expertises chez des clients variés et de se former en continu. J'ai été attiré par la "PALOITude" qui prône la bienveillance, le partage, l'engagement et le fun auprès de ses collaborateurs.

J'ai démarré mon aventure PALO IT par une mission BI pour une startup spécialisée dans le retail de luxe, avec pour objectif la mise en place d'une plateforme d'analytique pour ses clients, en passant par le design de cette solution, son prototypage et son industrialisation. La mission a duré 6 mois.

En attendant ma prochaine mission (tel un James Bond !), j'ai l'opportunité de contribuer sur des projets internes de la société. En novembre dernier, l'ensemble des collaborateurs Asie ont été conviés à un merveilleux team building à Phuket pendant lequel nous avons eu la possibilité d'échanger autour de l'avenir de la société. Depuis, je contribue au lancement de deux projets ambitieux : la "PALO IT Academy" et le "PALO IT Innovation Lab". En parallèle, PALO IT me propose régulièrement d'assister à des conférences pour me former et faire de la veille sur les tendances technologiques. J'ai d'ailleurs eu l'occasion de me pencher sur des technologies innovantes comme la Blockchain et d'aborder de nouvelles approches, comme l'idéation.

Des conseils à donner aux futurs expatriés ?

En conclusion, en arrivant à Hong Kong, je vous conseillerai de rester patient (en effet, certains processus sont bien plus longs qu'en France, comme obtenir un accès Admin sur une machine qui peut prendre 6 semaines !) et humble (en effet, les locaux n'ont pas forcément la même formation que les ingénieurs français, comme par exemple les classes préparatoires qui nous donnent un avantage considérable en maths et en méthodologie de gestion de projets). S'adapter au lifestyle Hongkongais est simple : c'est d'ailleurs une très bonne porte d'entrée pour démarrer une carrière professionnelle en Asie. Le pays est très occidentalisé (pas moins de 25 000 français y sont d'ailleurs implantés, mais essayez de ne pas rester qu'entre français !). Par contre, attendez-vous a payer 100 HK$ (12) pour 200 grammes de gruyère !

Pour finir, voici mes recommandations aux futurs expatriés :

- Se constituer un bon groupe d'amis sur place !
- Dire oui à tout, surtout la première année de votre arrivée ! Ne serait-ce que pour découvrir de nouvelles activités. Goûter des pieds ou tripes de poulet ? Oui ! Une randonnée sur une île perdue ? Oui ! Une soirée dans un maison abandonnée au fin fond de la forêt ? Oui !

Mais surtout : persévérez, car l'expatriation, ça vaut clairement le coup. D'ailleurs, PALO IT recrute de nouveaux consultants à Hong Kong, mais aussi en France, à Singapour et à Mexico !

ANECDOTE

Un de de mes anciens collègues avait refusé pendant un moment de travailler avec moi. J'ai compris après quelques semaines que c'était à cause d'un email que j'avais envoyé avec notre manager en copie, dans lequel je mettais en évidence une petite erreur de développement, ce qui était trivial pour moi. Mais pour lui, j'avais remis en cause ses compétences.

AUTRE PETITE ANECDOTE

L'humour au travail est un autre exemple de différence culturelle. L'autodérision et l'ironie ne sont pas tout le temps comprises. Exemple simple : il n'est pas rare pour moi de dire "Tu as raison, je suis bête" lors d'une discussion. Ne le faites surtout pas dans un cadre asiatique, vos collègues ne comprendront pas pourquoi vous vous insultez vous-même !

L'autre point qui pose parfois problème avec les locaux est la barrière de la langue. L'anglais n'est ni ma langue maternelle, ni la leur (qui est le cantonais !). Dans certain cas, cela génère des frictions voire des problèmes à cause d'un manque de maîtrise de l'anglais. Un développeur compétent et bourré de talent peut sembler idiot s'il ne comprend pas ce qu'on lui dit.

NE RATEZ AUCUN NUMÉRO

Abonnez-vous !

PROGRAMMEZ!
le magazine des développeurs

Nos classiques

1 an ..... 49€*
11 numéros

2 ans ..... 79€*
22 numéros

Etudiant ..... 39€*
1 an - 11 numéros * *Tarifs France métropolitaine*

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros

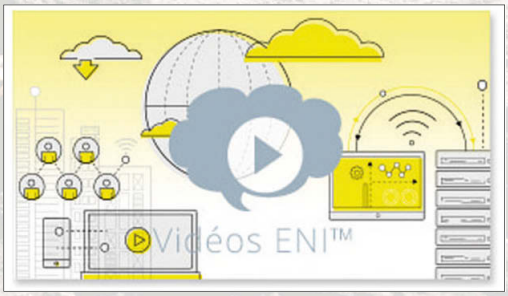
Souscription uniquement sur
www.programmez.com

Option :
accès aux archives 10€

Nos offres d'abonnements 2017

1 an ..... 59€
11 numéros + 1 vidéo ENI au choix :
- Big Data avec Hadoop
- Framework Spring

(Valeur de la vidéo de 29,99 à 59,99 €)



2 ans ..... 89€
22 numéros + 1 vidéo ENI au choix :
- Big Data avec Hadoop
- Framework Spring

(Valeur de la vidéo de 29,99 à 59,99 €)

Offre limitée à la France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ **Abonnement 1 an :** 49 €
☐ **Abonnement 2 ans :** 79 €
☐ **Abonnement 1 an Etudiant :** 39 €
Photocopie de la carte d'étudiant à joindre

☐ **Abonnement 1 an :** 59 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Big Data avec Hadoop
☐ Vidéo : Framework Spring

☐ Mme ☐ M. Entreprise : ..... Fonction : .....

Prénom : ..... Nom : .....

Adresse : .....

Code postal : ..... Ville : .....

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : ..... @ .....

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture

*** Tarifs France métropolitaine**

PROG 210
Offre limitée, valable jusqu'au 30 septembre 2017

Mobiles et tests sur mobiles

Bernard Homès
PDG **TESSCO sas**
Fondateur et ex-président du Comité Français des Tests Logiciels
(bhomes@tesscogroup.com)
Une expérience de 33 ans dans la qualité et les tests logiciels. Depuis 2000, il exerce en qualité de Consultant Senior pour le compte d'entreprises renommées dans la banque, l'aéronautique, le spatial et les télécommunications. Il intervient dans de nombreuses conférences et s'implique dans de nombreuses organisations professionnelles.

Les applications « mobiles » s'exécutent sur des terminaux mobiles et se déclinent d'une part en applications mobiles natives, développées spécifiquement pour être accédées par des équipements « mobiles » (p.ex. : tablette ou smartphone), d'autre part en applications qui ont été initialement conçues pour être accédées via un navigateur sur un support « mobile ». Du point de vue de l'utilisateur, il n'y a que peu de différences entre ces deux approches, même si les applications natives sont généralement mieux optimisées et mieux intégrées dans l'équipement mobile afin de fournir des expériences utilisateur plus riches.

Les terminaux mobiles

Par « terminaux mobiles » nous couvrirons les téléphones mobiles basiques (« feature phones » en anglais), non-intelligents et les téléphones à mi-chemin entre téléphones basiques et smartphones accédant au web via un navigateur. Il y a également les smartphones et tablettes, et autres équipements conçus spécifiquement pour l'utilisation mobile (montres, lunettes, etc.) permettant d'accéder à certaines applications, ainsi que des équipements à sécurité critique (p.ex. : équipements médicaux tels lecteurs de glycémie, pompes automatique d'injection d'insuline, etc.).

Typologie des applications mobiles

Comme les applications traditionnelles (sur PC) les applications mobiles peuvent être natives : elles sont alors livrées avec le mobile ou téléchargeables par celui-ci, et spécifiques au terminal mobile en question. Conçues pour interagir avec tous les composants matériels ou logiciels présents sur la plateforme (appareil photo, GPS, accéléromètre, haut-parleurs, micros, vidéo, etc.), ces applications peuvent subir des mises à jour, qu'il faut télécharger et installer sur la plateforme. Ces applications ne sont généralement pas portables sur d'autres plateformes.

Les applications mobiles peuvent aussi être basées sur des navigateurs mobiles et accessibles via des URL. Ces applications ne fonctionnent pas en l'absence d'un réseau et n'interagissent qu'avec un nombre restreint des composants présents sur la plateforme. Généralement, les mises à jour ne sont pas nécessaires car la logique est principalement sur les serveurs et non sur les plateformes mobiles. Ces applications sont d'ordinaire facilement portables.

Enfin, il y a des applications hybrides qui combinent les deux types d'applications précédentes : une interface native sur la plateforme, des services évolutifs sont présents sur les serveurs et des possibilités de mises à jour. Ces applications sont généralement facilement portables.

Utilisabilité, disponibilité et criticité

Les applications mobiles sont principalement à destination du grand public, donc les utilisateurs s'attendent à une utilisabilité intuitive, à une disponibilité à 100%, et à un temps de réponse adéquat ; quelles que soient la capacité et la puissance des réseaux disponibles. Les mobiles deviennent une extension du PC domestique ou de bureau.

Les applications mobiles – qui s'adressent principalement aux smartphones et tablettes – deviennent de plus en plus critiques dans notre vie de tous les jours. Il suffit de penser aux GPS et aux applications permettant de signaler à ses amis que l'on est en sécurité (utilisation de la géolocalisation), sans oublier les applications de gestion domotique, de paiement (achats sécurisés de biens ou de services), de réservation (p.ex. de billets de train ou d'avion), etc.

Lors de certains événements – festifs ou tragiques – on se rend compte que les plateformes mobiles sont souvent le mode de communication principal. Cela peut parfois saturer les réseaux et rendre les applications moins disponibles.

QUELS SONT LES CHALLENGES ?

Aux challenges des tests d'applications classiques, les tests d'applications mobiles ajoutent d'autres challenges liés spécifiquement aux applications mobiles :

- Un modèle économique basé sur un prix extrêmement bas et des ventes en volume ;
- Des configurations matérielles et logicielles extrêmement nombreuses et diverses ;
- Des aspects d'utilisabilité et de localisation (adaptation aux pays et aux langues utilisées) ;
- Des interactions avec d'autres applications ;
- Des éléments liés à la sécurité des données qui sont stockées sur ces équipements mobiles ;
- L'impossibilité d'avoir des exigences complètes vu l'évolutivité rapide des besoins.

Aspects économiques

Dans un modèle « traditionnel » sur PC professionnel, l'utilisateur était plus ou moins forcé d'utiliser un logiciel (soit il l'avait acheté, soit son utilisation était imposée par son organisation). Ce modèle évolue vers un paiement au service (SaaS) et à l'Open Source.

Dans un modèle « mobile », le concept économique est différent : les applications ont souvent un prix dérisoire (quelques euros) et il est facile pour un utilisateur – si l'application ne le satisfait pas – de changer d'application pour un produit concurrent. Nous voyons comme challenges particuliers :

- L'utilisabilité, critique pour fournir aux utilisateurs une expérience optimale ;
- Les temps de réponse très courts, quel que soit le réseau (y compris gestion des zones blanches) ou le type de connexion (wifi, 3G, 4G, etc.) ;
- L'utilisation de la géolocalisation pour offrir des services ciblés (p.ex. : publicité, ristournes, etc.), avec les aspects de confidentialité des données personnelles collectées ;
- Un besoin d'exhaustivité des tests et d'efficience des activités de test afin de trouver le plus de défauts avec l'effort le plus réduit ;
- Une croissance potentiellement extrêmement rapide du nombre d'utilisateurs en cas de succès. Ceci implique que les serveurs sur lesquels s'appuie l'application devront pouvoir s'adapter à une croissance très rapide.


Microsoft
Microsoft
experiences'17
L'événement de
l'intelligence numérique
Intelligence
artificielle
Confiance
numérique
New way
of work
Au Palais des Congrès de Paris
Mardi 3 octobre
JOURNÉE BUSINESS
Mercredi 4 octobre
JOURNÉE TECHNIQUE
#experiences'17
experiences17.microsoft.fr

Aspects techniques

L'explosion des configurations matérielles et logicielles sur lesquelles doivent fonctionner les applications est aussi un challenge important.
Parmi les challenges des tests logiciels liés spécifiquement aux applications mobiles, nous avons :

- Les applications mobiles sont fréquemment développées avec des méthodes agiles, voire DevOps, donc des livraisons fréquentes sont habituelles. Ceci nécessitera une automatisation de nombreux tests fonctionnels et techniques et l'intégration des divers niveaux de test au sein d'une stratégie de test complète et cohérente (un challenge majeur) ;
- La compatibilité des applications mobiles sur un grand éventail de plateformes (matérielles et logicielles) devrait être géré par la mise en place d'environnements de tests multiples, voire virtualisés ;
- Vu les changements fréquents de matériel, la portabilité des applications et de leurs données doit être assurée afin de faciliter les évolutions matérielles ;
- Certaines fonctionnalités (p.ex. l'accéléromètre ou la géolocalisation), nécessaires pour l'application (p.ex. pour détecter des chutes) peuvent être difficiles à tester.

Aspects d'utilisabilité

Comparé à un ordinateur et son duo clavier/souris, un terminal mobile permet des interactions beaucoup plus variées. L'écran tactile est un exemple évident, mais il est également possible d'utiliser la voix, les gestes, la reconnaissance d'image, voire le rythme cardiaque. Outre les interactions entre utilisateur et terminal, il faut gérer les interactions entre les applications : si vous êtes en voiture à l'approche d'une intersection, que votre application GPS vous indique de tourner et qu'un appel entrant vous parvient simultanément, quelle est l'application prioritaire ? Le GPS ou le téléphone ?
Il faut également envisager une utilisation de l'appareil au-delà des frontières. La continuité du service devra être assurée sur des réseaux différents. Sans oublier les cas où l'équipement est corrompu ou compromis, que ce soit par accident (p.ex. : perte ou chute), ou par une action malveillante (p.ex. : vol ou infection par un virus (cf. aspects de sécurité et de confidentialité ci-après).

Exhaustivité des exigences

Dans les applications mobiles encore plus que pour les applications traditionnelles, la combinatoire des exigences sera immense : d'une part nous avons les exigences fonctionnelles, mais nous aurons aussi toutes les exigences non-fonctionnelles – dont nombre ne seront pas écrites – comme les performances et les temps de réponse, la compatibilité, la sécurité, etc.
Le challenge ici est que de nombreuses exigences seront considérées comme implicites et seront probablement détectées tardivement.

Aspects de sécurité et de confidentialité

De plus en plus nous pouvons nous attendre à avoir des applications à sécurité critique utilisant des composants mobiles. Dans ce cas, les standards applicables devront être respectés et des preuves de complétude de couverture devront être fournies. Cela implique souvent l'utilisation de systèmes d'exploitation sécurisés.
Souvent ces systèmes ne sont considérés sécurisés que jusqu'à ce qu'une faille soit découverte. Certains éditeurs ont fait de cet aspect « sécurité » un argument de vente.
L'aspect de confidentialité est plus difficile à garantir :

- En ce qui concerne la géolocalisation, les mobiles émettent continuellement des informations de localisation ; entre autres chaque fois qu'ils passent d'un relais à un autre. Si la géolocalisation (le GPS) est activée la position géographique du mobile sera disponible ;
- En ce qui concerne la confidentialité des données (p.ex. : paiements dématérialisés), des messages et des conversations, à l'exception de certaines applications chiffrant les données, les informations sont principalement transmises en clair, donc non sécurisées.

Autres aspects de sécurité à envisager :

- Habilitation des fonctionnalités autorisées : éléments fonctionnels habituels et gestion des conditions d'erreur. Confirmation du rendu de l'application sur une large gamme de plateformes et de résolution d'écrans ; et bon fonctionnement des composants physiques (p.ex. : caméra, GPS, vidéos, microphones, accéléromètres ou gyroscopes, etc.) et des méthodes d'appel et de transmission de messages ;
- Prévention des fonctionnalités non autorisées (p.ex. l'envoi de données par un logiciel malveillant doit être prohibé). Certaines applications malicieuses envoient des données confidentielles par SMS (p.ex. : SMS Premium surtaxés), ou collectent des informations confidentielles, fournissant ainsi un point de départ pour des attaques informatiques (p.ex. social engineering).
- Limitation des permissions au strict minimum nécessaire pour l'application. Ici le challenge est de s'assurer que seules les fonctionnalités décrites dans l'EULA(1) sont utilisées. Ces fonctionnalités ne sont pas obligatoirement malicieuses, mais il est important de s'assurer que les applications utilisées n'autorisent pas des niveaux de permission inutiles ;
- Protection des données sensibles, à la fois en termes de confidentialité, mais aussi en termes d'intégrité. Ceci s'applique aux données sauvegardées de façon interne, mais sur les composants amovibles (cartes SD, etc.) et via les interfaces (p.ex. API). La protection des données sensibles peut nécessiter l'utilisation d'algorithmes de chiffrement des données pour un stockage chiffré sur les supports ;
- Dépendances limitées et justifiées envers des librairies externes approuvées ;
- Utilisation de réseaux ou d'applications non approuvés, ou interactions (p.ex. via Bluetooth ou clés USB) avec des systèmes physiques non approuvés ;
- La perte ou le vol des équipements, ou leur compromission (p.ex. via du phishing).

COMMENT TESTER TOUT CELA ?

La manière de tester les applications sur mobiles se traduira par la conception d'une stratégie de test basée sur les risques où seront étudiés tous les risques habituels (fonctionnels, délais de mise sur la marché, etc.) mais nous devrons nous focaliser sur certains éléments spécifiques.

Stratégie de test

Comme pour les applications traditionnelles, la stratégie de test doit couvrir simultanément :

- De l'analyse statique, manuelle (revues et inspections) ou automatisée des exigences et livrables utilisés (p.ex. code source, architecture, etc.), ainsi que l'analyse des vulnérabilités ;
- Du test fonctionnel et des tests non-fonctionnels (p.ex. : tests de performances, sécurité, etc.).

Les techniques de test usuelles (partitions d'équivalence, valeurs limites, cas d'utilisation, combinaisons, etc.) pourront être utilisées dans chacun des environnements, à chacun des niveaux (pour les développements séquentiels) ou lors de chaque itération (développements

(1) EULA: End User Licence Agreement

agiles et DevOps). Cependant les environnements de test ont des caractéristiques particulières qu'il faudra gérer.

Environnements de test

Il est peu réaliste d'envisager l'acquisition d'un exemplaire de chaque composant (smartphone ou tablette) sur lequel l'application doit fonctionner, puis de concevoir une automatisation spécifique. De nouveaux terminaux mobiles apparaissent et il faut ajouter les investissements en logiciels ou en connexions aux réseaux mobiles.

Des services permettent actuellement de simuler – p.ex. : dans le cloud – de nombreuses configurations matérielles et logicielles. Une recherche rapide permet de trouver des solutions de virtualisation, d'émulation et de simulation de plateformes mobiles.

La répartition statistique des types de mobiles et de tablettes est très volatile et évolue selon les utilisateurs. Certaines entreprises qui avaient il y a quelques années une place prépondérante dans les entreprises ne l'ont plus actuellement. Il faut donc connaître les types de mobiles utilisés par la cible de clientèle pour l'application.

Performances et Montée en charge

Les utilisateurs de terminaux mobiles s'attendent à des performances rapides, quelle que soit la capacité du réseau et le nombre d'utilisateurs simultanés. Les tests de performances devront donc mesurer les temps de réponse avec des bandes passantes différentes, voire avec des interruptions plus ou moins longues des connexions. Simuler ce type de réseau est un challenge important.

Devront être mesurés entre autres les aspects suivants :

- Durée de lancement de l'application (sur la plateforme mobile) ;
- Délais de l'interface utilisateur ;
- Présence ou absence d'indicateurs visuels en cas de performances dégradées ;
- Usage des ressources au sein de la plateforme mobile ;
- Durée de chargement des pages (p.ex. applications basées sur navigateurs).

Si le succès commercial est au rendez-vous, la charge sur les serveurs augmentera rapidement. Il sera nécessaire de tester la capacité des serveurs à traiter un nombre très important de connexions simultanées, et la capacité d'évolution de cette architecture. Les visions optimistes des services marketing peuvent être complètement dépassées par la réalité du terrain.

Sécurité

Comme pour les systèmes plus traditionnels, la sécurité des applications mobiles nécessite une spécialisation des acteurs (y compris des testeurs) et commence à être prise en compte.

De nombreuses sources sont disponibles pour développer des tests plus exhaustifs et plus complets. Parmi celles-ci nous pouvons citer le NIST (www.nist.gov National Institute of Standards and Technology) aux USA, et l'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) en France. L'ISTQB a aussi développé une certification Avancée en test de sécurité.

DES SOLUTIONS ?

Le test d'applications mobiles comporte les mêmes challenges que le test traditionnel :

- Délais réduits, charges de travail importantes, configurations nombreuses et variées, réutilisabilité des tests, besoins d'automatisation et de tests exploratoires manuels, etc.
- Nécessité de remontée des informations pertinentes et fréquentes à destination de toutes les parties prenantes (hiérarchie, clients et utilisateurs),
- Mesure continue de l'efficacité et de l'efficience des équipes avec des métriques (DDP, DRE(2), etc.) pour garantir l'amélioration continue des tests.

À cela viennent s'ajouter des challenges spécifiques aux terminaux mobiles :

- Interactions nombreuses avec des composants physiques internes ou externes ;
- Nombre de configurations matérielles et logicielles très important ;
- Communications sur des réseaux dont la bande passante et la latence sont variables ;
- Sécurisation des données, sur les terminaux mobiles et confidentialité des échanges ;
- Gestion de la publicité interactive associée à la géolocalisation.

Ceci implique que les testeurs doivent avoir des compétences techniques et méthodologiques particulières, qu'il est possible d'acquérir via des formations spécifiques.

Utilisation des métriques et mesures

Les métriques et mesures applicables aux tests des applications mobiles doivent fournir des informations sur la qualité du produit développé, pour prendre des décisions en temps utile (p.ex. : augmenter ou mieux cibler l'effort de test selon les résultats précédents).

Les développements étant principalement agiles et les livraisons continues, il peut être utile d'ajouter aux métriques traditionnelles des informations utiles au marketing de l'application :

- Nombre total de téléchargements (indique l'intérêt envers l'application et le nombre maximal d'utilisateurs simultanés) ;
- Nombre d'utilisateurs (nombre de personnes qui utilisent réellement l'application) ;
- Nouveaux utilisateurs (nombre d'utilisateurs utilisant l'application pour la première fois, à comparer au nombre d'utilisateurs) ;
- Fréquence, profondeur et durée des visites (montre l'intérêt et la fréquence d'utilisation) ;
- Taux de rebond (nombre de personnes qui téléchargent l'application, l'utilisent une seule fois et ne l'utilisent plus ensuite).

Ces mesures fournissent aux administrateurs et sponsors des informations précieuses dans le cadre d'un DEVOPS qui irait jusqu'à la livraison aux utilisateurs.

Certifications spécifiques

Les tests sur applications mobiles existent depuis de nombreuses années. Une formalisation s'est développée avec :

- CMAP testing (Certified Mobile Application Testing Professional) qui provient d'un groupe de travail sous l'égide d'iSQI. Ces certifications de testeurs sur mobiles existent depuis 2013 et comportent aussi les tests de performances sur mobiles et l'automatisation de tests sur mobiles.
- CMT (Certified Mobile Tester) qui propose depuis 2015 des certifications sous l'égide de l'ASTQB (comité Américain de l'ISTQB).
- L'ISTQB qui propose des certifications de niveau avancé en test de sécurité des applications.

(2) DDP : Defect Detection Percentage, DRE : Defect Removal Efficiency

Rendre les développeurs heureux en entreprise, c'est possible

Partie 1

Cyril Lakech
Cyril Lakech est développeur et leader technique chez AXA France où il a notamment pour mission d'animer la communauté des développeurs.

Romain Linsolas
Romain Linsolas travaille chez Société Générale comme leader technique d'une équipe élaborant les outils de développement d'applications Web pour toute la banque d'investissement.

Reconnaissons-le, le développeur jouit d'une situation très enviable : il est courtisé de toutes parts. Le nombre d'offres d'emploi dans le monde du développement peut faire rougir de nombreux autres secteurs d'activité. De l'autre côté du miroir — des entreprises donc — il devient ainsi compliqué de réussir à recruter les bonnes personnes, puis de les conserver au sein de leurs effectifs. Voici quelques pistes pour les entreprises qui souhaitent recruter et garder les meilleurs développeurs.

Investir dès le recrutement

C'est au cours du recrutement que l'entreprise donne une première impression au candidat, il est donc primordial que celle-ci soit bonne, sans quoi ce dernier vous ignorera tout simplement et passera à autre chose. Il apparaît qu'un mauvais recrutement est la deuxième cause d'échec d'un projet — la première étant son inutilité — alors autant prendre ce sujet avec le sérieux qu'il mérite. Il s'agit donc de savoir adapter le processus de recrutement aux besoins des développeurs. En voici quelques incontournables :

- Partager la vision et les objectifs de l'entreprise à moyen et long terme permet au candidat de comprendre le contexte de l'entreprise et les grands enjeux à venir. Il est tout aussi nécessaire de partager les valeurs premières de l'entreprise, celles auxquelles on doit se raccrocher quand on doute, celles qui portent l'état d'esprit des collaborateurs au quotidien. Il est vital que le candidat se retrouve dans ces valeurs et consente y adhérer.
- Décrire le cadre de travail et la façon dont s'organisent les équipes entre elles : quelle est l'ambiance de travail, quelles méthodes sont appliquées, quel est le fonctionnement classique des projets. Le candidat doit être en capacité d'imaginer son travail au quotidien dans l'entreprise.
- Aborder le sujet du salaire de manière ouverte. Il existe un marché dont les statistiques sont publiées régulièrement. Le salaire proposé doit être en adéquation avec ce marché et avec la mission. Cela semble évident, pourtant le sujet est généralement évoqué tardivement, et souvent de façon peu directe.
- Pour intéresser le développeur, rien de mieux que de parler du code, des outils, des langages et des technologies les plus répandus ou à l'étude. Les souhaits du candidat doivent correspondre aux pratiques de l'entreprise. Si l'entreprise pratique massivement les revues de code et les pratiques agiles, assurez-vous que les candidats acceptent ce genre de pratiques qui peuvent être déroutantes pour certaines personnes.
- Les *soft skills* sont un autre point sur lequel il apparaît important d'insister lors de l'entretien. Au-delà des compétences techniques, de bonnes qualités humaines sont tout simplement indispensables. Concrètement, un développeur passera peu de temps à écrire du code, il sera bien souvent occupé à échanger et à collaborer avec d'autres personnes. D'où l'importance de s'assurer au cours des entretiens que le candidat dispose bien de ces aptitudes.
- On pourra agrémenter le processus de recrutement avec un test technique ou du *pair programming* — c'est-à-dire de développer avec l'assistance d'une personne de l'équipe — afin d'avoir une mesure du niveau de maitrise technique du candidat. Ces épreuves sont parfois redoutées, mais sont généralement appréciées par les candidats, ravis d'être évalués par un pair sur des cas concrets.

Il est donc important d'impliquer les développeurs directement dans le processus de recrutement, afin de les engager et de les responsabiliser. Après tout, ce sont eux qui vont travailler avec les nouvelles recrues, autant s'assurer directement de leur compatibilité professionnelle. Rencontrer des développeurs durant ce processus est toujours bien perçu par les candidats, mais cela ne va pas sans inconvénients. Cela impose par exemple aux développeurs de l'équipe de passer du temps à réaliser ces entretiens, à en faire un suivi. On peut le voir comme du temps perdu — car non imputé sur le projet pour lequel il travaille — mais sur le long terme, l'entreprise est gagnante : un bon recrutement étant un véritable investissement sur l'avenir.

On l'a évoqué, le marché est aujourd'hui très favorable aux développeurs. Ces derniers sont de fait très sollicités par les entreprises en mal de talents. Si certaines d'entre elles vont jouer la carte de l'originalité pour sortir du lot, il ne faut pas tomber dans les travers des offres farfelues, voire risibles, où l'on cherche des profils tout simplement impossibles. « Je recherche un ninja du digital en mode rock-star DevSecOps et évangéliste avec 10 d'expérience sur Kotlin pour un grand groupe international à l'esprit startup et à taille humaine ». Non seulement ce genre d'offre ne trouvera jamais preneur, mais elle aura surtout le mérite d'irriter le potentiel candidat et de vous décrédibiliser complètement. À proscrire absolument !

Éviter aussi de faire coder un exercice ou un algorithme sur un tableau blanc en ayant la même rigueur qu'un compilateur. Non seulement cela ne mène à rien mais cela risque aussi de faire fuir votre candidat. Le tableau blanc reste une bonne idée s'il est utilisé pour démontrer la façon dont le candidat réfléchit et amène sa solution, éventuellement avec l'aide de pseudo-code. Si votre souhait est de voir le candidat à l'œuvre avec du véritable code, il est préférable de se tourner vers des solutions telles que Codingame, actuellement utilisée chez AXA France et Société Générale.

Et pourquoi ne pas proposer au candidat de s'immerger directement dans l'équipe pendant quelques heures, de l'impliquer dans une séance de revue de code ou de *pair programming*. Arriver à faire passer ses premiers tests unitaires au vert avec le candidat, c'est une excellente manière de démarrer une collaboration efficace.

Enfin, la qualité des rapports humains entre les membres de l'équipe est essentielle. Avoir dans son équipe une fameuse *rock star* du JavaScript, mais qui a un caractère invivable risque de poser plus de problèmes que d'apporter des solutions au sein de l'équipe. Proposer d'aller boire un verre peut être une manière informelle de s'en assurer, et de mettre à l'aise le candidat. Dernier point crucial sur le recrutement : la transparence et la franchise sur l'entreprise et l'équipe qu'il pourrait rejoindre. Si, après quelques semaines ou quelques mois, il s'aperçoit d'une quelconque supercherie, il sera déçu, démotivé et il y a de fortes chances qu'il quitte le navire. Et comme évoqué au début de l'article, un mauvais recrutement est extrêmement coûteux pour l'entreprise.

Proposer un vrai plan de carrière

Il arrive que les développeurs s'ennuient et finissent par venir travailler simplement pour remplir le frigo. Ce n'est souhaitable ni pour eux ni pour l'entreprise. Il convient de leur offrir la possibilité de construire leur avenir afin de les fidéliser au sein de l'entreprise. Nombreux sont les développeurs qui se sentent désengagés, déresponsabilisés dans leur travail, alors qu'ils pourraient être épanouis dans leur carrière — ce qui aurait également l'avantage non négligeable d'améliorer leur productivité.

Pour fidéliser le développeur dans l'entreprise, il faut lui proposer une expérience durable dans laquelle il va pouvoir évoluer dans le temps. Il n'est plus question de considérer le métier de développeur comme une passerelle avant le métier de manager ou chef de projet. Aujourd'hui, de multiples alternatives existent.

- Faire évoluer son niveau d'expertise technique, avec de multiples paliers tels que novice, junior, sénior et enfin expert.
- Profiter de sa capacité d'influence et de leadership, par exemple pour occuper des postes d'évangélistes.
- Parfaire sa maîtrise des technologies complémentaires, telles que le développement Web, *back-end* (avec Java, .Net, etc.), mobile, etc.
- Développer sa maîtrise des domaines métiers de l'entreprise.
- S'orienter vers des rôles d'architectes techniques ou fonctionnels.

Toutes ces évolutions du profil de développeur peuvent être clairement identifiées et valorisées dans l'entreprise, ce qui donne aux développeurs une vision long terme de leur carrière. Cela permet aussi à l'entreprise de voir dans quels domaines et technologies il y a une carence ou une abondance de compétences au sein de ses effectifs.

L'une des manières les plus sûres pour obtenir l'attention des développeurs est de leur proposer de vivre des expériences extraordinaires. Il faut que leur quotidien soit varié, en alternant des projets stratégiques avec des projets innovants, ponctués parfois des projets plus classiques. Varier les plaisirs et casser la routine permettra aux développeurs de se sentir valorisés, et davantage motivés.

Un autre aspect à ne pas négliger est la formation, l'apprentissage. Pour paraphraser Damien Cavaillès : « Développeurs, apprendre est notre métier ». Pour rester employable, un développeur doit apprendre sans cesse et suivre l'évolution des technologies. Si l'on veut garder les développeurs et en attirer d'autres, il faut leur proposer une carrière où ils auront l'assurance d'être en mesure d'apprendre continuellement, de s'améliorer. Laisser du temps aux développeurs pour qu'ils puissent produire tout en apprenant est important à leurs yeux, sans quoi ils iront apprendre ailleurs, ou le feront en cachette sur le temps imputé aux projets sur lesquels ils travaillent. Dans de telles conditions, personne n'en sort gagnant !

À bien y regarder, il n'y a que deux possibilités.

- Soit l'entreprise investit sur la formation des développeurs. Cela représente un coût — certains diront plutôt un investissement, mais présentera l'avantage de créer un cercle vertueux où l'entreprise, les développeurs ainsi que les clients et utilisateurs s'y retrouveront au vu de l'amélioration notable de la qualité des productions.
- Soit l'entreprise n'investit pas dans la formation de ses développeurs. Ce choix stratégique va nécessairement impliquer à moyen terme une difficulté accrue à conserver les meilleurs éléments au sein des effectifs, et à rendre le recrutement encore plus difficile.

De leur côté, les développeurs doivent apprendre à apprendre, ce qui n'est pas aussi facile qu'il n'y parait. Une manière intéressante et efficace de le faire est d'apprendre à enseigner. Partager sa passion est souvent une caractéristique reconnue d'un bon développeur, mais cela a aussi de nombreux autres atouts. Hubert Sablonnière (*@hsablonniere* sur Twitter), un développeur Web et orateur apprécié, a déclaré que de « donner des cours et des présentations a changé ma vie professionnelle. C'est un terrain d'auto-apprentissage inépuisable qui stimule les connaissances et savoir-faire liés à la technique mais aussi aux relations humaines. » Quand les développeurs émettent ce souhait, il est primordial de les encourager à donner des cours, des présentations, des *Brown Bag Lunches*, tout en valorisant ce travail.

Créer un environnement stimulant

La carrière du développeur étant désormais entre de bonnes mains, il faut maintenant s'affairer à lui proposer un environnement de travail qui soit propice à la concentration et à l'efficacité. L'open-space, très présent dans les entreprises, n'est clairement pas adapté à ces contraintes : bruyant, source de déconcentration et de stress, généralement dépourvu d'âme, on lui préfèrera les bureaux flexibles. Ces lieux viennent casser la routine créée par le bureau traditionnel en offrant des espaces adaptés aux besoins du moment. On adoptera ainsi les petits espaces intimes pour suivre une conférence téléphonique sans être dérangé. On privilégiera les petites zones confortables pour les meetings en petit comité. L'intérêt des espaces de détente est aussi minimisé par les sociétés, alors qu'il s'agit là d'un lieu important, véritable exutoire pour le développeur qui aura passé une matinée entière à dénicher le bug dans son programme ! C'est aussi un lieu idéal pour souder les équipes, où chacun pourra mieux découvrir ses partenaires professionnels. Si cette configuration des lieux est souvent la signature des géants du Web ou des startups, les grandes entreprises commencent elles-aussi à s'y mettre, y voyant clairement un investissement incontournable et un atout séduction auprès de potentiels candidats à recruter. À titre d'exemple, Société Générale a ouvert en 2016 ses nouveaux locaux à Val de Fontenay — « Les Dunes » — dans cet état d'esprit.

L'espace de travail est lui aussi souvent mal considéré. Il ne suffit plus d'une simple chaise et un coin de table pour travailler. Les développeurs pouvant passer près de dix heures par jour à leur poste, il devient primordial de leur assurer un confort certain, et de veiller à leur santé. À cet égard, on pourra opter pour le « bureau debout », offrant la possibilité de travailler aussi bien assis que debout et de changer régulièrement de posture de travail. Il va de soi qu'une chaise de qualité ira de pair avec ce bureau. Le télétravail est encore mal établi dans la plupart des grandes entreprises, alors que le métier du développement informatique s'y prête parfaitement ! C'est pourtant un critère de plus en plus important lorsqu'un candidat cherche un nouveau poste, et pour certains d'entre eux, un critère éliminatoire. Nous n'allons pas retracer ici les nombreux avantages

que procure ce mode de travail, mais ces derniers sont encore plus mis en exergue pour les métiers du développement : travailler chez soi offre un plus grand confort en partie grâce au calme, une meilleure concentration et donc une augmentation de la productivité. Bien entendu, cela nécessite de la part du télétravailleur une certaine rigueur organisationnelle. Il est primordial d'être en mesure de s'isoler, de créer un environnement spécifique afin de limiter les perturbations extérieures, mais également pour marquer — physiquement et psychologiquement — une véritable frontière entre le milieu professionnel et celui personnel.

Arrêtons-nous maintenant sur une pièce maîtresse et vitale du développeur : la machine. Elle est fréquemment source de crispations : trop lente, trop limitée, peu adaptée au développement, mal configurée... les critiques ne manquent pas ! Hormis l'aspect de la sécurité qui est cruciale pour les grandes sociétés, c'est très souvent le prix qui sert d'excuse pour ne pas fournir un matériel de meilleure qualité. Pourtant un calcul très simple permet de montrer qu'il n'en est rien : si l'on considère qu'un prestataire est facturé 600€ par jour — ce qui correspond à un tarif tout à fait honnête sur la région parisienne, et si ce dernier vient à perdre une heure quotidiennement à cause de son matériel peu performant, alors la perte induite peut avoisiner les 15 000€ par an ! Admettons-le, même une excellente machine n'atteindra jamais de tels tarifs.

Cette même logique s'applique aux logiciels mis à disposition des développeurs. Il est de plus en plus contre-productif d'imposer un logiciel unique à toutes ses équipes, on préfèrera opter pour une solution standard, tout en laissant le choix à chacun d'opter pour ses outils privilégiés. Après tout, un développeur qui maîtrise parfaitement un outil risquera d'être perdu, et donc moins productif, si on vient à lui en imposer un autre.

Joël Spolsky, célèbre entrepreneur et co-fondateur de Stack Exchange et Trello, écrivait en 2000 le très connu "Test de Joël" (1) permettant à un candidat d'évaluer le niveau de qualité et de maturité de l'équipe de développement qu'il pourrait rejoindre. Les règles huit et neuf de ce test résument parfaitement ce que nous venons de dire au cours de ce chapitre :

- Les programmeurs ont-ils un environnement de travail calme ?
- Disposez-vous des meilleurs outils que vous puissiez vous payer ?

Il ne faut donc pas faire l'économie d'un bon environnement de travail pour le développeur, ce dernier vous le rendra par une productivité et une motivation accrues !

Le Software Craftsmanship à la rescousse

Un développeur veut pouvoir s'épanouir dans son travail et apprendre en permanence. Comment l'amener à cet épanouissement ? Comment lui permettre de s'améliorer continuellement et qu'il puisse prendre conscience qu'il devient meilleur chaque jour ? La réponse à ces questions tient en deux mots, le *Software Craftsmanship*, qui pourrait se traduire par le développement artisanal.

L'agilité semblait être la solution à tous les maux du développement logiciel, et s'il est vrai que cela a considérablement amélioré la situation, cela ne suffit plus. L'agilité est souvent utilisée pour améliorer l'efficacité des livraisons et le résultat est que l'on se retrouve à livrer plus vite des applications parfois non fonctionnelles, défectibles et difficiles à maintenir. Le *Software Craftsmanship* ne se définit pas comme un remplaçant de l'agilité, mais plutôt comme une extension de celle-ci, en rappelant que les logiciels doivent être bien conçus, et que chaque nouveau développement doit clairement apporter de la valeur au produit final. Il rappelle également que les développeurs font partie d'une communauté de professionnels qui s'entraident, et que la relation avec le métier n'est pas simplement basée sur le mode client - fournisseur mais doit s'appuyer sur un partenariat vertueux.

Le *Software Craftsmanship* porte en soi une valeur importante, celle de l'idée que l'on doit être fier du travail réalisé collectivement et de l'apprentissage permanent. Concrètement, de nombreuses pratiques peuvent être mises en œuvre, telles que le *TDD* (*Test Driven Development*), le *BDD* (*Behavior Driven Development*), la mise en application des préceptes du *clean code* (code propre) avec ses principes *DRY* (*Don't Repeat Yourself*), *KISS* (*Keep It Simple, Stupid*) ou encore *SOLID* (acronyme représentant les cinq principes de bases de la programmation orientée objet). L'envie ici n'est pas d'ajouter de multiples acronymes barbares sur son CV, mais plutôt de savoir adapter sa façon de travailler pour produire du code plus clair, plus expressif et à la maintenance facilitée. Il faut beaucoup de persévérance pour créer une culture d'entreprise qui soit proche des valeurs et des pratiques du *Software Craftsmanship*. Ce processus est long, mais à travers lui vous apprendrez beaucoup, la qualité de vos livrables s'améliorera, et il sera alors difficile de revenir en arrière tant les bénéfices sont nombreux ! Richard Branson, fameux entrepreneur et fondateur de Virgin, a parfaitement résumé le challenge des entreprises sans se limiter au monde de l'informatique : « formez vos collaborateurs afin qu'ils puissent partir, mais traitez-les suffisamment bien pour qu'ils n'aient pas envie de le faire ».

Travailler mieux, mais pourquoi ?

L'objectif du *Software Craftsmanship* est de produire du code avec une qualité toujours accrue, et cela est motivé en partie parce que la responsabilité du développeur est de plus en plus souvent engagée. Aujourd'hui, la mise en production se fait souvent par les équipes mêmes qui ont développé le produit, ou, à défaut, en coordination directe avec eux. Sans cela, il ne faut pas espérer les responsabiliser ; en cas de défaut, le développeur pourrait alors se décharger sur les opérateurs qui ont mis en production sa réalisation. Cette approche, mise en exergue par le mouvement *DevOps*, permet ainsi aux développeurs de voir les fonctionnalités dans leur ensemble, de l'idéation à l'utilisation du produit par les utilisateurs finaux. Si cette approche permet de donner plus de responsabilité aux développeurs, elle offre aussi plus de visibilité à leur travail, ce qui ne manquera pas de les rendre fiers de ce qui a pu être accompli pour en arriver là.

Si désormais les développeurs réalisent de belles applications qu'ils mettent eux-mêmes en production, il reste un dernier point : savoir donner du sens à leur travail. Participer à une énième application de type *CRUD,* ou une application sans intérêt, risque de lasser les équipes. La première cause d'échec des startups est de créer une application qui ne sert à rien, ou qui n'est utile à personne ! N'oublions pas non plus l'un des grands attraits de cette profession, à savoir qu'il est possible et aisé de travailler dans des domaines extrêmement variés : du domaine bancaire au domaine scientifique, en passant par les voyages, les services, aucun secteur n'y échappe. Ne pas réussir à intéresser le développeur au métier sous-jacent, c'est prendre le risque de voir ce dernier quitter votre société par lassitude ou fatigue, afin d'aller vivre des aventures passionnantes ailleurs. •

Suite et fin au prochain numéro

[1] https://fr.wikipedia.org/wiki/Test_de_Joël

Un **serveur Web** et un **site Internet** dans l'Internet des Objets (IoT)

Christophe Villeneuve
Consultant IT pour Ausy, Mozilla Rep, auteur du livre "Drupal avancé" aux éditions Eyrolles et auteur aux Editions ENI, PHPère des elePHPants PHP, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Drupagora...

Les objets connectés sont de plus en plus présents et de plus en plus puissants. Ils ouvrent aussi de nombreuses possibilités, comme avoir un serveur Web portable, accompagné de son site Internet. Ainsi, la technologie facilite l'opération pour permettre lors d'un événement ou d'un hackathon d'avoir un ensemble de développeurs avec un serveur dédié portable connecté.

Lors d'événements, de hackathons ou de défis, où Internet n'est pas toujours disponible, il est toujours utile d'avoir un serveur Web à portée de main pour répondre aux attentes des participants, ou d'un site d'actualité. Pour cela, nous utiliserons les objets connectés pour faciliter l'installation et le paramétrage avec des solutions libres.

Le kit

Un serveur Web, ne se limite pas qu'au Cloud ou à un serveur mutualisé distant, il peut être physique près de vous et même tout le temps avec soi. Pour posséder son serveur portatif, qui sera disponible à tout moment, vous devrez investir quelques dizaines d'euros au niveau du matériel avec l'utilisation de logiciels libres.

Matériels

- Carte : Raspberry Pi 2 ou Pi 3
- Mémoire : 1 Go ou +
- Carte SD : 16 Go ou +
- Connexion Internet : par câble Ethernet ou WiFi

Le matériel facultatif :

- Ecran HDMI
- Clavier
- Souris

Logiciels

Pour réaliser un serveur Web, nous utiliserons différents paquets libres. Tout d'abord, Raspbian sera le système d'exploitation libre et gratuit, basé sur Debian, optimisé pour fonctionner sur Raspberry PI.

Ensuite, notre serveur utilisera les logiciels suivants :

- Apache : 2.4.10
- PHP 7
- MariaDB 10

Pour terminer, nous utiliserons Drupal pour réaliser notre site Internet.

Préparation

Préparons notre serveur.

Etape 1 : formatage

Nous formatons notre carte SD en FAT 32. Il existe de nombreux outils, mais nous utiliserons ceux-ci :

- Sous Linux : Gestionnaire de partition ;
- Sous Windows (et macOS) : SDFormatter ;
- Sous Mac : RPi-sd card builder.

Bien entendu, il est possible de formater en ligne de commande ou avec le logiciel de formatage natif des OS.

Etape 2 : installation image

Nous installons l'image du système. Pour cela, nous téléchargeons la dernière version complète (https://raspbian-france.fr/telechargements/) appelée Raspbian Jessie pour notre matériel. Après le téléchargement et la décompression du fichier, vous utiliserez le logiciel suivant :

- Pour Windows : Win32DiskImager
- Pour Mac : Disk imager
- Pour linux : En ligne de commande

Pour installer Raspbian sur votre carte SD à partir de Linux, nous effectuons l'opération suivante :

```
sudo dd bs=1M if=chemin_vers_le_img_de_raspbian of=/dev/votre_carte
```

Quelle que soit, la méthode utilisée pour installer l'image sur la carte SD, vous l'insérer dans votre Raspberry Pi. Connectez l'alimentation. Et le système démarre.

Identification réseau

Comme il s'agit d'un serveur portable, il faut lui attribuer des droits serveurs. Pour cela, nous lui ajoutons à partir du mode console les lignes suivantes :

```
sudo systemctl enable ssh.service
sudo apt-get install openssh-server
sudo openssh –version
sudo touch /boot/ssh
sudo reboot
```

Ici, nous installons et activons un serveur SSH. Notre machine serveur est maintenant sur le réseau. A partir de votre ordinateur, vous vous connectez en SSH à l'IP de la machine (ici 192.168.0.36) :

```
$ ssh pi@192.168.0.36
```

Les identifiants par défaut sont :

```
login : pi
password:  raspberry
```

[1]

Installation

L'installation d'un serveur Web sur une Pi s'effectue principalement en mode terminal c'est à dire à distance. Sinon, il est toujours possible de le faire directement sur le serveur si vous possédez l'ensemble des options pré-requises.

Pré-installation

Pour bénéficier des dernières versions des logiciels, vous mettez à jour manuellement la version de Raspbian à partir des derniers dépôts :

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get upgrade
```

On met à jour la liste des fichiers disponibles dans les dépôts APT présents dans le fichier de configuration. Ensuite Upgrade met à jour tous les paquets installés vers les dernières versions. Maintenant nous sommes prêts pour installer notre environnement.

Installation de Apache

Tout d'abord, nous installons Apache, qui sera notre serveur Web. Pour cela nous effectuons l'opération suivante :

```
sudo aptitude install apache2
```

Après l'installation de notre serveur, nous pouvons le vérifier en lançant notre navigateur avec l'adresse suivante :

```
http://192.168.0.36
```

[2]

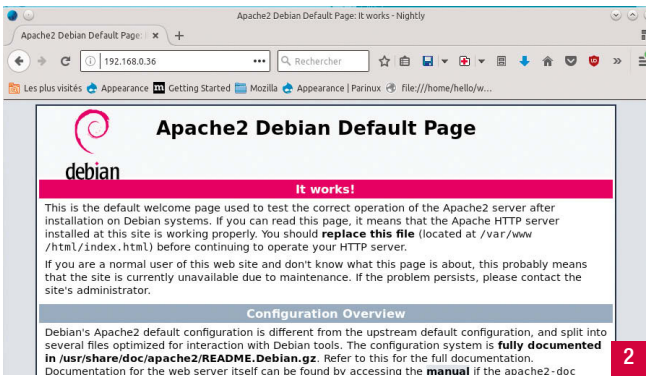
Nous devrions arriver sur cette page qui montre bien que le serveur fonctionne correctement. Nous préparons notre serveur :

```
cd /etc/apache2/sites-enabled
sudo nano 000-default.conf
```

Nous ajoutons les lignes suivantes :

```
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

<Directory /var/www/html>
  Options Indexes FollowSymLinks MultiViews
  AllowOverride All
  DirectoryIndex index.php
  Order allow,deny
  Allow from all
</Directory>
```

1



2

Nous sauvegardons et redémarrons le service Apache

```
sudo /etc/init.d/apache2 restart
```

Installation de MariaDB

Comme nous avons mis à jour notre système d'exploitation Raspbian, nous bénéficions des derniers dépôts disponibles pour MariaDB. Pour cela, nous effectuons les opérations suivantes :

```
sudo aptitude install mariadb-server mariadb-client
```

L'opération a installé la base de données serveur et client avec la configuration par défaut. Lors de l'installation, il est demandé de saisir un mot de passe pour le compte ROOT, que vous devrez renseigner.
Pour rendre exploitable le serveur de base de données, il faut initialiser le mot de passe 'root', ainsi que des privilèges anonymes... C'est pourquoi nous apportons un minimum de sécurité supplémentaire avec la commande suivante :

```
sudo mysql_secure_installation
```

Un certain nombre de questions sont posées :

- Enter password for user root: <-- Entrez votre mot de passe ROOT
- Change the password for root ? <-- y
- Remove anonymous users? <-- y
- Disallow root login remotely? <-- y
- Remove test database and access to it? <-- y
- Reload privilege tables now? <-- y

Nous vérifions que la base de données est correctement installée avec la fonction suivante :

```
mysql -u root -p
```

[3]

Installation de PHP

Nous pouvons maintenant installer le langage PHP et différentes extensions associées pour qu'il fonctionne avec le serveur :

```
sudo aptitude install php7.0 libapache2-mod-php7.0 php7.0-curl php7.0-gd
php7.0-fpm php7.0-cli php7.0-imap php7.0-json php7.0-mcrypt php7.0-mysql
php7.0-opcache php7.0-xmlrpc libapache2-mod-php7.0 php7.0-xml
```

Nous redémarrons le serveur Apache pour que les différents ajouts soient pris en compte :

```
sudo systemctl restart apache2
```

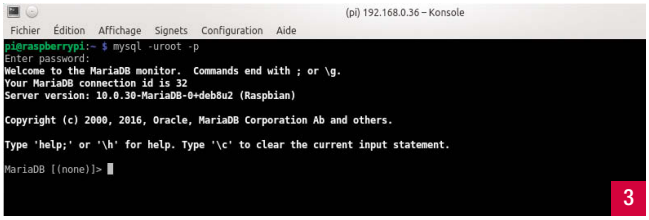
Vérification

Maintenant que nous avons installé l'ensemble des paquets

Vérification version PHP

```
apache -v
```

[4]



3

```
php -v
```

[5]

Complément

Pour améliorer la configuration de notre serveur, nous activons quelques modules complémentaires indispensables.
Tout d'abord, nous activons le module mod_rewrite pour permettre la réécriture des URLs et nous redémarrons le serveur :

```
sudo a2enmod rewrite
sudo service apache2 restart
```

Ensuite, nous sécurisons notre serveur avec SSL. Pour cela, nous ajoutons le module SSL pour Apache, le virtualhost et redémarrons le serveur :

```
sudo a2enmod ssl
sudo a2ensite default-ssl
sudo service apache2 restart
```

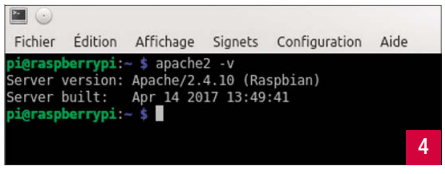
Nous en profitons pour attribuer un utilisateur différent du système d'exploitation, afin de permettre l'accès au serveur Web. Pour cela, nous faisons :

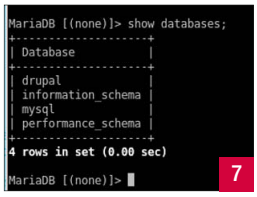
```
sudo adduser pi www-data
sudo chown -R pi:www-data /var/www/html/
sudo chmod 755 -R /var/www/html
sudo chmod g+s -R /var/www/html
```

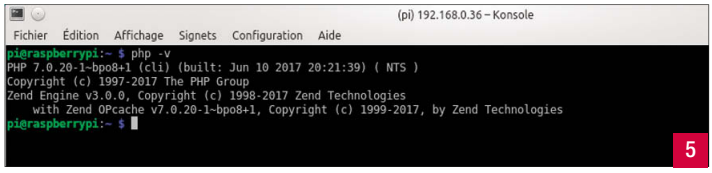
La première ligne ajoute l'utilisateur PI au groupe www-data, la deuxième ligne attribue à cet utilisateur la propriété du serveur Web. Les deux dernières lignes vont autoriser les utilisateurs membres du groupe www-data à pouvoir lire et écrire dans tous les fichiers se trouvant dans /var/www/html. Enfin, nous vérifions que notre espace Web est prêt à l'emploi avec la ligne suivante :

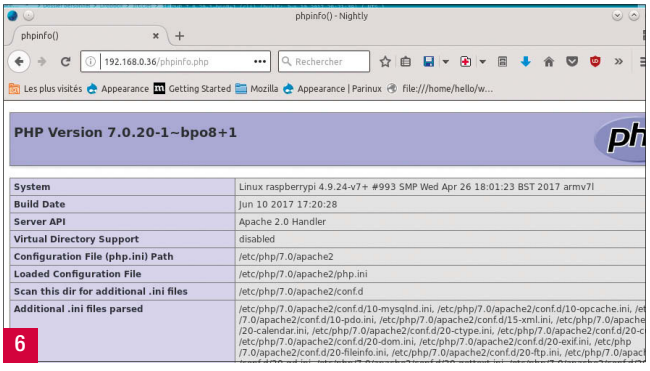
```
echo "<?php phpinfo();" >> /var/www/html/phpinfo.php
```

[6]









Drupal

Maintenant installons un site Internet, de la même façon que sur un serveur mutualisé ou dédié. Nous choisirons le CMS Drupal pour vous montrer une façon rapide d'avoir un site Internet disponible. Il existe différentes manières pour installer ce CMS de façon manuelle ou automatique. Tout d'abord, nous créons la base de données, appelée "drupal". Pour cela, nous faisons ceci :

```
mysql u root -p
```

Nous créons la base de données :

```
create database drupal;
```

Nous vérifions que celle-ci soit bien créé :

```
show databases;
```

[7]

Après avoir créé notre base, installons notre CMS. Pour cela nous chargeons notre dernière version 8 de Drupal, et nous décompressons l'archive.

```
cd /var/www/html
sudo wget https://ftp.drupal.org/files/projects/drupal-8.3.3.tar.gz
sudo tar -zxvf drupal-8.3.3.tar.gz
```

Pour lancer la procédure d'installation, nous retournons sur notre navigateur et nous tapons ceci :

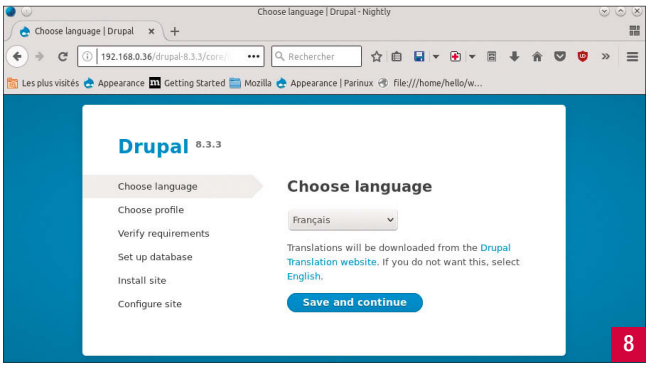
```
http://192.168.0.36/drupal-8.3.3
```

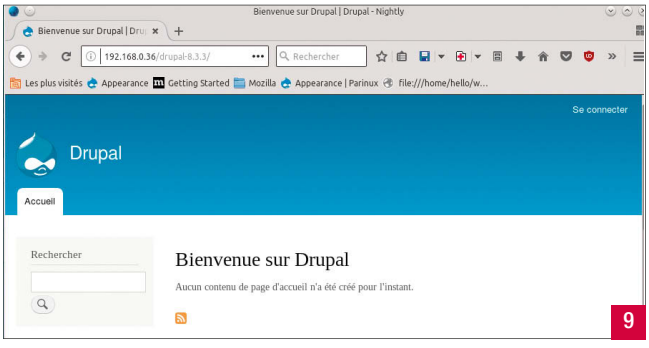
[8]

Vous suivez les différents écrans, et renseignez les différents champs. Lorsque l'ensemble des étapes a été effectué, vous obtenez ceci : [9]

CONCLUSION

Comme vous avez pu le voir il est très simple d'installer un serveur Web complet sur son Raspberry Pi et de créer des sites Web.
A vous de jouer !





« Blockchain-as-a-Service » et **Blockchain 2.0**

Igor Leontiev
Cloud Architecte
@VISEOGroup
MVP Azure

Vincent Thavonekham
@VISEOGroup
Microsoft Regional Director et MVP Azure

Microsoft Regional Director

Le « blockchain-as-a-service » (BaaS) offre de nombreux services dédiés à la Blockchain qui permettent aux développeurs et aux entreprises de développer rapidement des applications peu chères, avec de faibles risques. Nous obtenons ainsi des applications dites « décentralisées » (DApps pour Decentralized Applications), qui permettent aux sociétés de travailler ensemble sur des nouveaux cas d'usages parfois « disruptive » (i.e. très en rupture des concepts actuels), au travers de l'usage de plateformes Cloud.

Microsoft a mis à disposition en octobre 2015 un BaaS, qui fut talonné de près par IBM quatre mois après (Ref 1). Entre effets d'annonces et réalités, est-ce qu'il existe de véritables projets BaaS ? ou bien n'est-ce qu'un mythe ?
Suite à la levée de la confidentialité et de l'annonce publique de carnets d'entretien infalsifiables de RENAULT (Ref 2), cet article vous présente une partie de ce qui a été mis en place.
Sachant qu'en 2018, Microsoft nous prépare davantage de nouveautés, et cela sera l'occasion de publier un second article !

Historique et définition

Les premières chaînes de blocks sont apparues en 1991, et depuis, de nombreuses personnes parlent de Blockchain, sans parfois en comprendre les tenants et les aboutissants, ou n'ont qu'une vision partielle de cela, tant les niveaux d'expertises technique et fonctionnelle sont vastes ; nous débuterons donc par la définition anglaise que nous avons reprise en français, suivie d'une explication de texte.

Anglais : *« Blockchain – is a distributed database that is used to maintain a continuously growing list of records, called blocks. Each block contains a timestamp and a link to a previous block. A blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for validating new blocks. By design, blockchains are inherently resistant to modification of the data. Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks and a collusion of the network majority. Functionally, a blockchain can serve as "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way. The ledger itself can also be programmed to trigger transactions automatically. »*

Français : "La Blockchain est une base de données distribuée. Elle est utilisée pour maintenir à jour une liste grandissante d'enregistrements appelés 'blocks'. Chaque bloc contient un timestamp (i.e. une date/heure précise) ainsi qu'un lien vers le bloc précédent. Une Blockchain est typiquement gérée par un réseau de Peer-to-peer partageant tous le même protocole de validation des nouveaux blocs. Par nature, la Blockchain est intrinsèquement résistante à toute modification de la donnée. Une fois enregistrée, la donnée d'un bloc ne peut être modifiée rétroactivement sans altérer tous les blocs sous-jacents. A moins d'une puissance de calculs exponentielle et une corruption par entente illicite de 51% du réseau (Ref 3).

Fonctionnellement, une Blockchain peut servir de livre de registres qui enregistre efficacement les transactions entre deux tiers, et cela de façon vérifiable et permanente. Ce livre de registre peut être programmé afin de lancer des transactions automatiquement." En résumé, une Blockchain est une structure de données créée pour numériser des livres de registres. Une Blockchain est partagée sur un réseau distribué (peer-to-peer) comme eMule. Ce qui les rend quasi-indestructibles, contrairement à du stockage de données sur un ou des serveurs.

Etat des lieux en 2017

A ce jour dans Azure, il existe une Marketplace comportant 10 solutions partenaires offrant des services telles que l'infrastructure, la gestion des identités, et un environnement de développement intégré. On distingue les catégories suivantes :

- **Outillage** – pour développer sur les problématiques de Blockchain ;
- **Registres à un nœud** (Single node ledgers) – un réseau comportant seulement un nœud de Blockchain, habituellement conçu pour des besoins de développement ;
- **Registres multi-nœud** (Multi node ledgers) – solution entièrement préconfigurée avec un réseau d'entreprise complet.

Voici quelques exemples sur la Marketplace de la plateforme Azure : [1]
Les solutions se développent rapidement, et avant de débuter tout projet Blockchain, il est important de trouver celle en adéquation avec notre besoin. Ce qui nécessite diverses évaluations. Outre le fait que nous ayons retenu le template Ethereum pour cet article et pour le projet RENAULT, c'est également le choix le plus populaire pour construire des solutions tant publiques que privées. En effet, Ethereum bénéficie d'une communauté mondiale et bien développée, avec des outils comme Truffle et Nethereum, respectivement connus en tant que Frameworks de développement répandus, et une solution clef en main pour accéder au réseau Ethereum.
En outre, l'avantage principal d'Ethereum est qu'il permet de créer des « contrats intelligents », plus connus sous le nom de « Smart Contracts » programmables. Nous allons voir comment utiliser le Framework Truffle, ainsi que Solidity (une extension à Visual Studio). Enfin, un dernier avantage, est que pour nos développements et notre mise en production, les applications « Back-End » ainsi créées sont compatibles avec un déploiement rapide sur la plateforme Azure, malgré la lourdeur d'un réseau Ethereum en production.

Introduction à la conception d'applications "DApps"

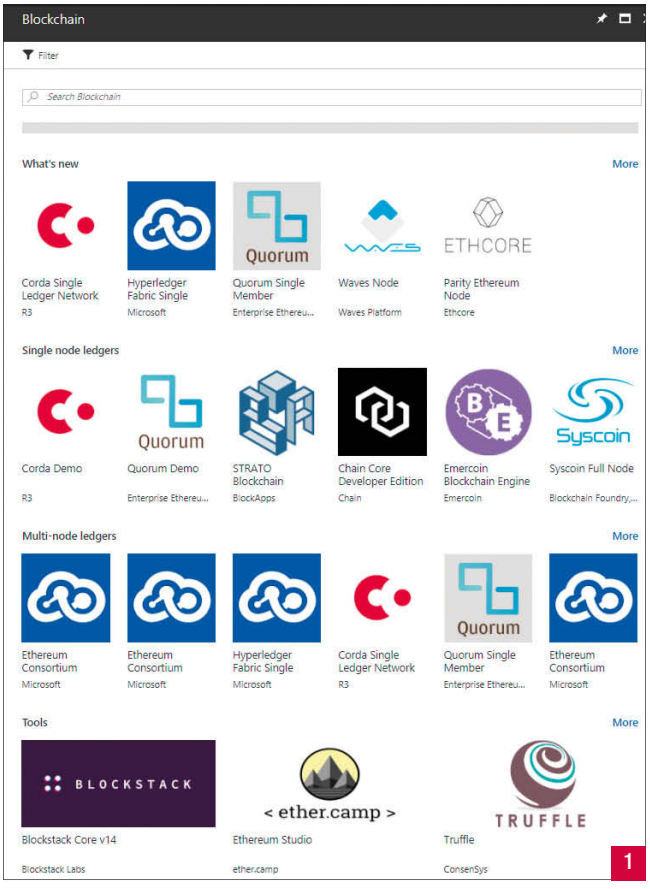
Pour concevoir une application DApps, des bonnes pratiques d'architecture sont nécessaires et sont répertoriées dans le GitHub de ConsenSys (Ref 4). Il faut avant tout s'interroger sur les types d'interactions avec les autres DApps (ex. échanges décentralisés ou non, …) ainsi qu'anticiper le cycle de vie de leur état (dans le sens « state ») au fil du temps. Dans certains cas, une certaine modularité peut être souhaitée lorsque des fonctionnalités exigent des mises à jour, sans pour autant recopier et réenregistrer tous les états dans la base de registre des DApps.

Voici les exemples repris du GitHub de ConsenSys :

- **Closed-system DApps** – forme la plus simple, avec un unique contrat qui intègre ses états et ses fonctions.
- **Closed-system DApp with modularity on the edges** – plus complexe, ce contrat peut être autonome mais va plutôt interagir avec les autres, afin de rajouter des fonctionnalités nouvelles, par rapport au cas précédent.
- **Multi-contract & Multi-state** – C'est le type de contrat le plus sophistiqué, avec des problématiques à traiter ventilées dans différentes fonctionnalités. Certains éléments du contrat ont une durée de vie différente d'autres parties.
- **Hub and Spoke** – Les contrats dit "Spoke" (en français ce sont des rayons de vélos) sont des contrats long-terme qui possèdent des états "statefull". Et les contrats "Hub" permettent des contrôles d'accès aux méthodes interagissant avec les "Spokes".

Illustrons cela avec le cas du "Closed-system DApp with modularity on the edges", qui semble être une bonne introduction, puisque ni trop simple, ni trop complexe. [2]

Dans Ethereum, cela s'implémente par deux éléments :



- Les Smart Contrats à déployer sur le réseau Ethereum Network (en fait, on va le déployer directement une VM Ethereum) ;
- Une application côté Client qui dialogue avec le réseau Ethereum au travers de web3.js.

C'est ce que nous allons détailler.

Les API JavaScript pour Ethereum - Web3JS

Il existe plusieurs langages de programmation pour écrire des Smart Contracts, citons notamment :

- **Solidity** – semblable au JavaScript, avec une extension de fichier .sol ;
- **Serpent** – écrit en C++, Serpent ressemble à du Python avec une extension de fichier .se ;
- **LLL**– se base sur une syntaxe LISP, mais avec du code proche de l'assembleur.

Dans tous les cas ces langages recherchent la haute performance et veulent manipuler des nombres astronomiques.

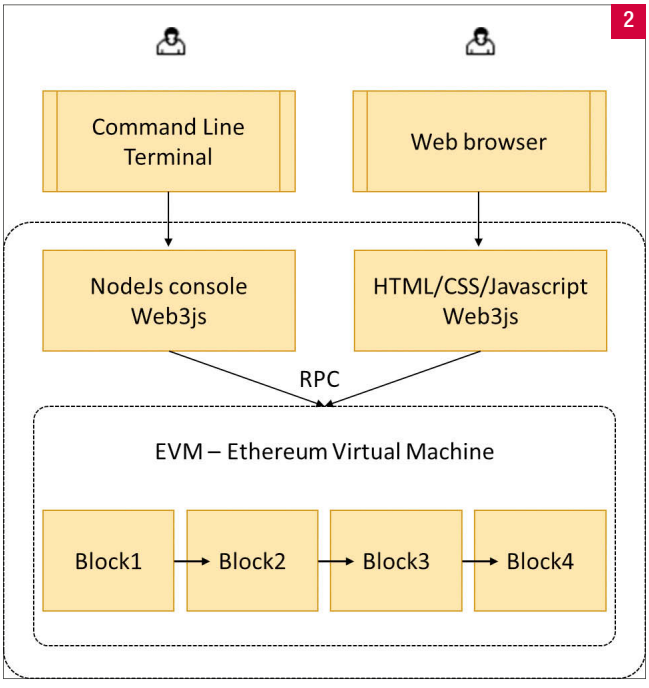
Pour l'heure, Solidity est l'un des langages les plus populaires et robustes ; Solidity sera utilisé dans cet article. A noter que Serpent (semblable au serpent python !) est l'un des premiers langages mais accuse des problèmes de sécurité (Ref 5).

Implémenter sa blockchain : préparation de l'environnement

Lorsque nous travaillons sur le sujet de la Blockchain, au-delà des aspects fonctionnels qui peuvent s'avérer très complexes, le déploiement d'une application sur un réseau Blockchain relève du défi. Cela est d'autant plus vrai avec des environnements de production, où des problématiques de sécurité et de performances sont intéressantes mais difficile à appréhender.

Pour exécuter nos tests vous allez pouvoir utiliser des machines locales de développement ; ultérieurement pour de la production nous pourrons utiliser les templates de Consortium BaaS. Dans les deux cas, cela nous permet de déployer rapidement dans Azure, car Microsoft gère une bonne partie de la complexité pour nous.

Nous allons utiliser du Linux qui fonctionne sous Windows 10 : « Linux



Subsystem on Windows 10 » (WSL). Il faudra le configurer d'une façon spécifique (Ref 6: Par défaut, cette fonctionnalité est désactivée « Compatibility layer for running Linux binary executables 'ELF format' natively on Windows 10).
Débutons par un Bash Ubuntu, puis installons WSL npm. Attention : pour installer npm, nous devons utiliser le repository official npm, car apt-get a une version très ancienne de npm.

```
curl - sL https: //deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt - get update - y && sudo apt - get upgrade - y
```

Ensuite, installons NodeJS :

```
sudo apt - get install - y build - essential python nodejs
```

Avant d'installer d'autres outils, nous devons mettre à jour npm et seulement après nous allons pouvoir installer Ethereum-TestRPC Truffle (pour réaliser des tests).

```
sudo npm install - g npm
sudo npm install - g ethereumjs - testrpc truffle
```

TestRPC – est un Node.js basé sur un client Ethereum pour développer et tester ses applications. Il se base sur ethereumjs pour simuler le comportement d'un client complet, permettant ainsi de développer des applications Ethereum bien plus rapidement. Il inclut aussi les fonctions RPC et les "features" (comme les évènements).
Truffle – environnement de développement intégrant un framework de tests et des pipelines pour Ethereum.
Enfin, achevons la préparation de la machine de développement en installant Geth qui est une commande en ligne pour exécuter un nœud complet d'Ethereum, qui sera implémenté en Go (Go Ethereum).

```
sudo apt - get install software - properties - common
sudo add - apt - repository - y ppa: ethereum / Ethereum
sudo apt - get update sudo apt - get install ethereum
```

Blockchain : construction d'un premier smart contract

Suite à l'installation des outils de développement, nous allons débuter la création d'un premier projet constitué d'un Smart Contrat et d'une interface utilisateur. Pour cela, basons-nous sur un repository truffle (Ref 7) qui inclut des contrats, des outils de migration, des tests, des interfaces utilisateur et un package pour construire son pipeline.
Pour créer ce projet, simplement saisir

```
truffle init webpack                                    [3]
```

Par défaut, truffle inclut un exemple de contrat (MetaCoin et ConvertLib) qui est un projet http://altcoins.com qui s'avère être une alternative de BitCoin, mais conçue avec Ethereum. Les Smart Contrats se trouvent dans le répertoire « contracts », et la description de la solution complète est :

- Contracts – répertoire avec les Smart Contracts ;
- Migrations – Ensemble de scripts de déploiement « managés » ;
- Test – Répertoire avec l'intégralité des tests ;

Note : Truffle va exécuter les tests uniquement à partir de fichiers dont l'extension serait : .js, .es, .es6, and .jsx, and .sol (les autres extensions seront ignorées)

- Truffle.js – Fichier de configuration du réseau, sur lequel nos Smart Contracts seront déployés ;
- FrontEnd – Fichiers du Front.

Analyse du Contract

Considérons le répertoire "contracts" où se trouve le fichier Metacoin.sol
La première variable du contract.

```
mapping (address => uint) balances;
```

est utilisée pour stocker le solde du compte de chaque individu (cela représente l'adresse de leur portefeuille électronique).
Ensuite, nous y retrouvons deux fonctions :

```
function MetaCoin() {
  balances[tx.origin] = 10000;
}
function sendCoin(address receiver, uint amount) returns(bool sufficient) {
  if (balances[msg.sender] < amount)  return  false;
  balances[msg.sender] -= amount;
  balances[receiver] += amount;
  Transfer(msg.sender, receiver, amount);
  return true;
}
```

Fonctionnement des fonctions : certains appels de fonctions peuvent être des Transactions ; les appels de fonctions qui changent l'état du contrat (modification de valeurs, ajout d'un enregistrement, etc.) sont des transactions et ont un émetteur implicite et une valeur.
A l'intérieur des accolades { from: __, value: __ } nous pouvons spécifier des fonctions web3.js pour envoyer des fonds à une fonction de Transaction d'un portefeuille. L'exemple précédent ajoute au solde du compte 10000 « Ether ».
Du côté de Solidity, nous pouvons récupérer ces valeurs avec msg.sender et msg.value. Ces deux fonctions sont des transactions implicites dans Solidity.
Certaines fonctions peuvent retourner uniquement la valeur des données stockées sur le réseau. Par exemple, les deux fonctions ci-dessous retournent le solde depuis le réseau :

```
function getBalanceInEth(address addr) returns(uint) {
  return ConvertLib.convert(getBalance(addr), 2);
}
```

```
root@INV050405:~/programmez# truffle init webpack
Downloading project...
Installing dependencies...
Project initialized.

  Documentation: https://github.com/trufflesuite/truffle-init-webpack

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test:           truffle test
  Build Frontend: npm run build
  Run Linter:     npm run lint
  Run Dev Server: npm run dev

Hint: Run the dev server via `npm run dev` to have your changes rebuilt automatically.

Make sure you have an Ethereum client like the ethereumjs-testrpc running on http://localhost:8545.

root@INV050405:~/programmez#
```

3

```
function getBalance(address addr) returns(uint) {
  return balances[addr];
}
```

Comprendre les tests truffle

Dans le répertoire "test", ouvrir le fichier TestMetacoin.sol :

```
contract TestMetacoin {
  function testInitialBalanceUsingDeployedContract() {
    MetaCoin meta = MetaCoin(DeployedAddresses.MetaCoin());
    uint expected = 10000;
    Assert.equal(meta.getBalance(tx.origin), expected, "Owner should have 10000
MetaCoin initially");
  }

  function testInitialBalanceWithNewMetaCoin() {
    MetaCoin meta = new MetaCoin();
    uint expected = 10000;
    Assert.equal(meta.getBalance(tx.origin), expected, "Owner should have 10000
 MetaCoin initially");
  }
}
```

Nous observons deux différences écrites selon les bonnes pratiques AAA (Arrange, Act, Assert). Leur but est de tester le Solde. A noter que cet exemple illustre cela de deux façons différentes, avec une partie Assert qui respecte le Standard JS de tests.

Déployer et tester les Smart Contract

Sur la machine de développement, nous allons ouvrir une nouvelle session Linux et lancer testrpc, qui, comme déjà évoqué, permet de tester le client Ethereum avec nos contrats sans avoir recours au véritable réseau Ethereum. Pour cela testrpc va créer un ensemble de comptes valides avec leur clef Private. Avec HTTPListener qui émule le véritable réseau Ethreum Network [4]

```
root@INV050405: ~
EthereumJS TestRPC v3.0.5

Available Accounts
==================
(0) 0xd0f3d5bbaa5174c681c373c646cdbef0b70cf937
(1) 0xc4cc7c8443b39ae341669d6e35676fbea6744a13
(2) 0x1bddba323fea1e3f551262048a37d084ae38c1b5
(3) 0xe1c44ad3c695443e894f5ac2fa33d703eba579f3
(4) 0x8221f39971e1b623a7250128160 4a9f657b72ea8
(5) 0xa59f1fcedce0dcbc4bb372afb215495397e1d923
(6) 0xd164717f3a50ffd8c1d88bea1d1b93d4935c8722
(7) 0x687147a99921aba06cefcb3edac21b3ab50711ad
(8) 0x63ad0b706de4a6b5efc92f61a4251a42b1f3f286
(9) 0x3841fa1ad815edab0bf27b375b74ce9a673165b9

Private Keys
==================
(0) bf47100345bb64284c884b8b45bc8431e4ce3044d04e9672d4e1d14df7f99ef1
(1) 292051a2c342d2ce5ceb033b4a19b4d1c45fd33faf1bc8d7c36ddc320c4852be
(2) fc3bc5e1517506179afa41e109f1c6048c97ab2e3477cded9a188f2b4bb2ef74
(3) 16eb6763e2ba45ae519b7e032793861fc42c9668197d79d9b201253d4ff6b781
(4) d134ad3cfd3dcbf9d91561e71f62e29c9b22e19d546c9430d747186bf2ade75a
(5) 3ac2a899da0e91aa76fc11d03ec71beb1a727dbeb07187a60304d49f674593fa
(6) 15a258525111c4a6c0aa3a19d38e9bf63a6b4c6112a0406b4b98825fe51fbe6f
(7) 12fa23d263f45796d5d759531bba06fc86271c0788824689c8f79aecc4254362
(8) 9d70219d5850ea4c8bae57c77f67f09126d1f19b10879c5ba42bd5e832cffe4c
(9) f425695661b04cd3c5781b8209ef67d9458b02ed83d2c23e032b1206fbf6ad6e

HD Wallet
==================
Mnemonic:      park special evolve luxury enjoy blind present area drift
 dog castle correct
Base HD Path:  m/44'/60'/0'/0/{account_index}

Listening on localhost:8545
```

4

Compilons et déployons nos Smart Contracts sur testrpc. Pour cela, retournons vers notre première fenêtre WLS

```
truffle compile
truffle deploy
```
[5]

Sur la seconde fenêtre WSL, nous pouvons voir les changements dans le réseau [6]

A présent, testons notre Smart Contract. Pour ce faire, utilisons la console qui fait partie intégrante de truffle framework :

```
truffle console
```

Puis, pour récupérer tous les comptes :

```
web3.eth.accounts
```
[7]

Nous pouvons ainsi vérifier que les réponses contiennent tous les comptes créés par testrpc

A présent, nous allons récupérer les références aux contrats déployés :

```
var metaCoin;
MetaCoin.deployed().then(function(deployed) {
  metaCoin = deployed;
});
```

```
root@INV050405: ~/programmez
root@INV050405:~/programmez/test# nano TestMetacoin.sol
root@INV050405:~/programmez/test# cd ~/programmez/
root@INV050405:~/programmez# truffle compile
Compiling ./contracts/ConvertLib.sol...
Compiling ./contracts/MetaCoin.sol...
Compiling ./contracts/Migrations.sol...
Writing artifacts to ./build/contracts

root@INV050405:~/programmez# truffle deploy
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  Migrations: 0x93f9284385fccf773dd876dc1ee5f6a27b2d837d
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying ConvertLib...
  ConvertLib: 0x1967fa4d4b2de69d283ba772f86c6eaea73f1179
  Linking ConvertLib to MetaCoin
  Deploying MetaCoin...
  MetaCoin: 0x0406c25de1e8b410be55759e539e5d191b4ea84d
Saving successful migration to network...
Saving artifacts...
root@INV050405:~/programmez#
```

5

```
root@INV050405: ~
  Transaction: 0xe5b0087d84c3d5fa04fe4863acc60e7bc8fd22e02c16ae49da1330f
1b0fd9371
  Contract created: 0x0406c25de1e8b410be55759e539e5d191b4ea84d
  Gas usage: 0x0371a4
  Block Number: 0x04
  Block Time: Tue Jul 11 2017 17:33:54 GMT+0200 (DST)

eth_newBlockFilter
eth_getFilterChanges
eth_getTransactionReceipt
eth_getCode
eth_uninstallFilter
eth_sendTransaction

  Transaction: 0x2627a74454bc5a3cd8234f54de6fa572ec36f5081b6491737326990
95e6ad0de
  Gas usage: 0x68ae
  Block Number: 0x05
  Block Time: Tue Jul 11 2017 17:33:54 GMT+0200 (DST)
```

6

Récupérons le Solde de notre contrat

```
metaCoin.getBalance.call(web3.eth.accounts[0])
```

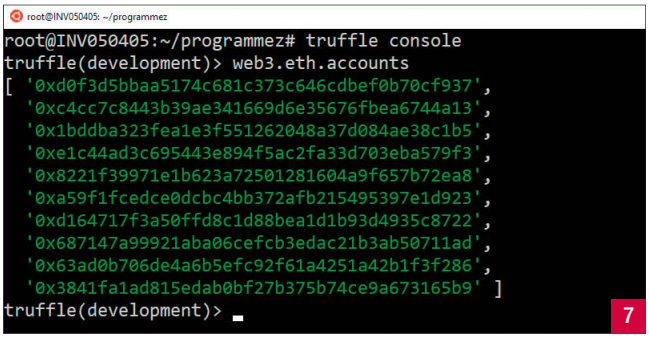
La Réponse doit être la suivante : [8]
Et enfin, testons les Smart Contract fonctionnellement en leur envoyant des pièces :

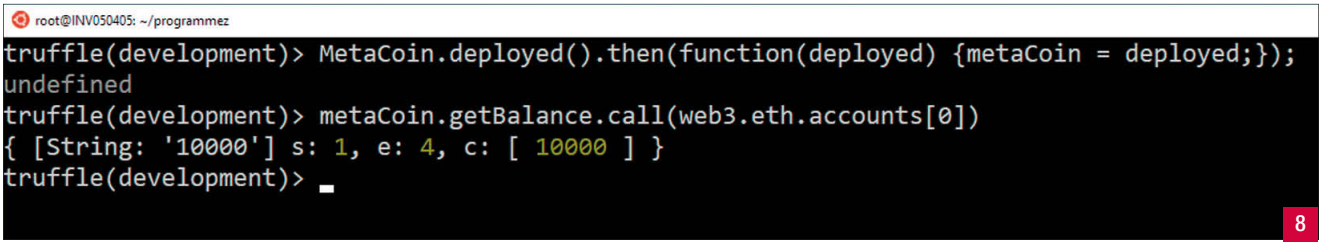
```
var account1 = web3.eth.accounts[1];
metaCoin.sendCoin(account1, 1000, {
  from: account0
});
metaCoin.getBalance.call(account0);
```

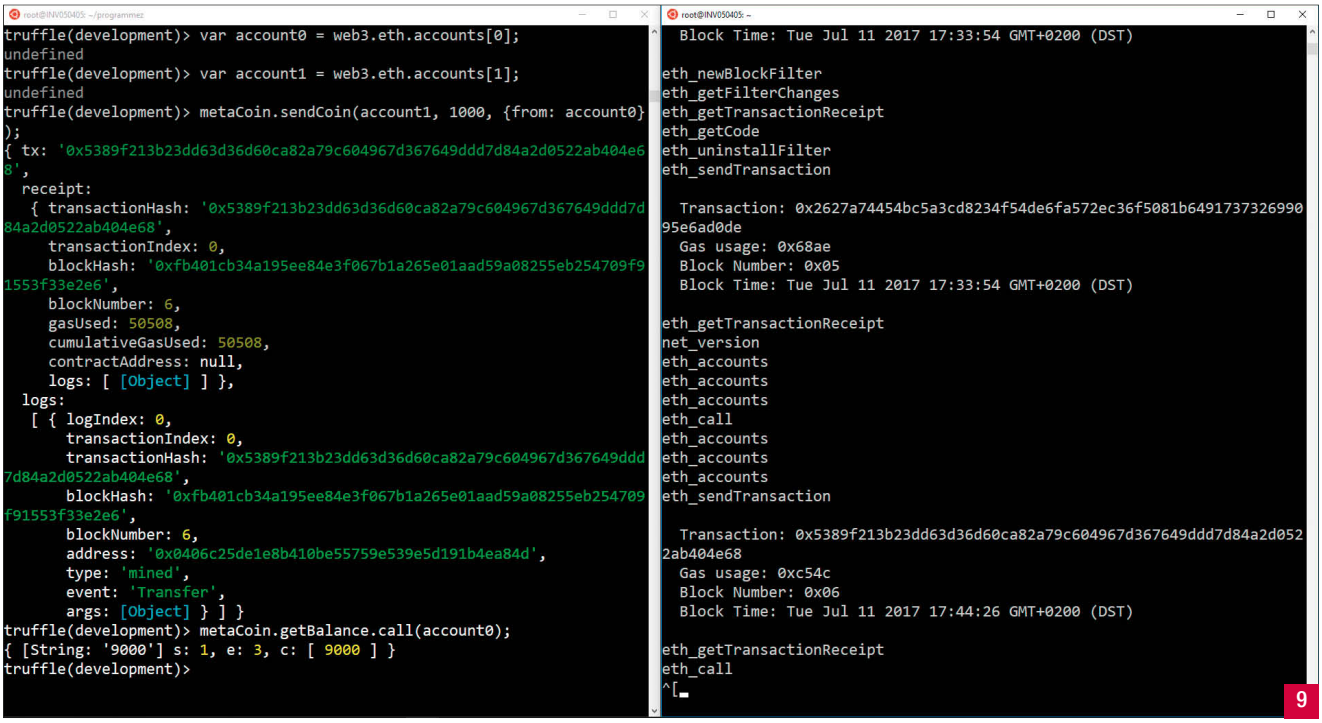
L'écran montre à gauche la Réponse, et à droite l'écran « rpc console » identifiée par son ID Test. [9]
A gauche, nous voyons bien que metaCoin.getBalance.call retourne bien 9000 « Ether ».
Cet exemple, se voulait simple fonctionnellement pour commencer à appréhender les difficultés liées à la Blockchain du fait de la forte traçabilité et de la sécurité en place, même sur des cas aisés. Cela montre du coup l'intérêt du BaaS, qui nous permet, à nous développeur, de gagner du temps et de la stabilité dans notre code. Ainsi nous-est-il possible de mieux nous concentrer sur les aspects fonctionnels souvent complexes du projet.

Références :

BaaS par IBM et Microsoft
https://www.nextinpact.com/news/98575-ibm-et-microsoft-misent-sur-blockchain-as-a-service.htm
Renault développe le carnet d'entretien du futur avec VISEO et Microsoft :
http://www.reparateur-carrossier-auto.fr/actualites/renault-developpe-carnet-dentretien-futur
Attaque des 51% https://fr.wikipedia.org/wiki/Attaque_des_51%25
Bonnes pratiques d'architecture DApps
https://github.com/ConsenSys/Ethereum-Development-Best-Practices/wiki/Dapp-Architecture-Designs
Serpent un langage pour les Smart Contracts d'Ethereum en déclin
https://www.coindesk.com/one-of-ethereums-earliest-smart-contract-languages-is-headed-for-retirement
Linux Subsystem on Windows 10 :
https://msdn.microsoft.com/fr-fr/commandline/wsl/install_guide
Exemple de code Smart Contract :
https://github.com/trufflesuite/truffle-init-webpack







Apple rejoint WebVR

Début juillet, Apple annonçait son ralliement à WebVR ! C'est une demie surprise depuis la présentation officielle d'ARKit, en attendant la partie matérielle dédiée qui devrait être dévoilée sur les prochains iPhone. Mais pour Apple, il s'agit maintenant d'étendre au-delà de sa technologie. WebGL sur Safari serait donc un pas important. Plusieurs développeurs pommés travailleront au WebVR Community Group (W3C). Cela peut signifier l'arrivée de matériels dédiés et le besoin d'avoir du contenu notamment sur le navigateur. Le support passera sans doute par WebKit dans les prochains mois.

en bref

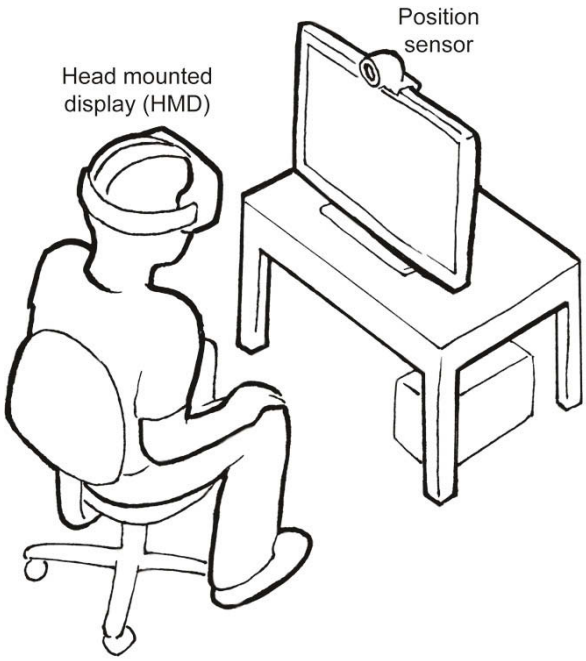
HTC vive baisse de 200 €

Le nouveau **Hololens** sera disponible courant 2019, peut-être dès 2018. Ce nouveau casque incorporera un processeur dédié à l'intelligence artificielle, obsession des géants technologiques depuis quelques mois. Le SoC utilisé actuellement par Hololens changera aussi car Intel ne le fabriquera plus.

Oculus Rift a eu droit à une baisse de prix avec une promotion spéciale été : le casque + 2 manettes vendus à 449 euros. Offre malheureusement arrêtée.

Un HTC Vive sans fil est disponible en Chine. Il est équipé d'un processeur Snapdragon 835. Le casque autonome, certes moins puissant mais aussi moins cher, est une tendance forte. Facebook prépare le sien et Google l'a annoncé pour les prochains mois, le fameux VR Daydream.

WEBVR : L'AVENIR DE LA VR PASSE AUSSI PAR LE NAVIGATEUR



WebVR est un standard ouvert pour proposer une expérience de réalité virtuelle dans le navigateur. Le but : proposer cette expérience à tout le monde. Mais pour cela il faut un matériel et un navigateur. WebVR est une API JavaScript.

Actuellement, WebVR est compatible avec Cardboard, Daydream, Samsung Gear VR, Oculus Rift, HTC Vive, Playstation VR et Windows Holographic. Mais il est possible d'avoir une certaine expérience VR sans matériel dédié mais dans ce cas, on perd toute l'interactivité. Le navigateur doit être compatible WebGL. Ce support varie beaucoup d'un navigateur à un autre et sur l'OS où tourne le navigateur. Actuellement, aucune compatibilité sur macOS. Edge est le moins riche en support excepté la partie Microsoft.

L'API a été développée par Mozilla. Pour le moment, elle est toujours expérimentale. Elle gère le matériel (accélération, acquisition, position, etc.), le rendu stéréoscopique. Comme il s'agit d'une API orientée Web et JavaScript, la programmation ne va pas vous surprendre.
Mais WebVR est loin d'être l'unique API VR sur les navigateurs. On peut citer ReactVR qui peut en faire ou encore le framework a-frame. Ce dernier permet de faire du WebVR et est supporté par Mozilla.

	Firefox Chrome	Android	Chromium	Edge
VIVE	x	-	x	-
Oculus	x	-	x	-
Windows réalité mixte	-	-	-	x
Cardboard	?	x	selon OS	?

ARKit : annonce d'un succès sans précédent ?

Il faut bien le reconnaître, la présentation lors de la conférence développeur d'Apple, la WWDC, de la technologie ARKit a provoqué un engouement auprès des développeurs iOS, et des autres que nous avions rarement vu pour une technologie de réalité mixte / virtuelle / augmentée. Et tout cela sans aucun matériel spécifiquement dédié, en attendant l'annonce courant septembre des nouveaux iPhones et de la sortie d'iOS 11 qui incluera ARKit. De très nombreuses démos et apps ont été développées depuis juin dernier, voici quelques exemples :

- Un robot se promène dans la rue par Duncan Walker : https://www.youtube.com/watch?v=cSxhGvJyaNo
- Un des exemples les plus spectaculaires, une stargate quand on veut ! site : https://www.youtube.com/watch?v=vRahhbXZT9I
- Un impressionnant clip ARKit façon A-ha : un des meilleurs exemples pour nous. https://www.youtube.com/watch?v=ZBdRAdSosv4
- Le studio Wingnut AR parle de son travail sur ARKit et de sa démo présentée durant la WWDC : https://www.youtube.com/watch?v=NOdTyVvRMgY

Depuis août, Unreal Engine 4 supporte officiellement ARKit (version bêta pour le moment). Ce premier support était immédiatement disponible sur Github.
Bref, une techno à suivre de très près par celles et ceux qui veulent s'investir dans ces technologies. Jusqu'à présent, la réalité virtuelle sur mobile n'a pas été un franc succès malgré la communication intense de Samsung depuis quelques semaines.

Angular 4, nom de code : invisible-makeover

• Daniel DJORDJEVIC,
Project Lead Web

• William KLEIN,
Consultant **Infinite Square**

• Sébastien OLLIVIER,
Tech Lead Web, MVP
Auteurs du livre : Angular, Développez vos applications web avec le framework JavaScript de Google

INFINITE SQUARE http://blogs.infinitesquare.com/

Il y a quelques mois, l'équipe de Google responsable du développement d'Angular a publié une nouvelle version de son Framework : Angular 4 - nom de code invisible - makeover (ou changement invisible en français). Comme son nom le laisse présager, la majorité des changements apportés à cette version sont invisibles pour les développeurs que nous sommes et interviennent sur les mécanismes internes. Angular 4 est donc plus rapide et produit des applications plus petites. Voyons ensemble les principaux ajouts de cette version.

Mais où est passé Angular 3 ?

Avant de rentrer dans le vif du sujet, revenons quelques instants en arrière. En 2009, Google publie la première version de son célèbre Framework AngularJS. En 2016, Google publie une version totalement réécrite de son Framework adaptée aux nouvelles contraintes du développement Web : Angular 2. Ce changement de version permet d'illustrer la cassure réalisée. En 2017, Google publie une nouvelle version : Angular 4. Mais où est passé Angular 3 ?

Le module de routing @angular/router d'Angular 2 est en version 3. Pour éviter de se retrouver avec des versions différentes pour chacun de ses modules (et éviter le syndrome du "je fais de l'Angular 2, avec le router version 3, les animations version 1 et les formulaires version 6"), l'équipe a décidé de réaligner les versions de tous les modules Core à la version 4. La version Angular 3 n'existe donc simplement pas puisque tout a été réaligné en 4.

Angular 4 est-elle une version totalement réécrite d'Angular 2 ?

Angular 4 ne représente pas une version totalement réécrite d'Angular 2, comme a pu l'être Angular 2 avec AngularJS. Angular 4 représente une nouvelle version majeure d'Angular 2, apportant un ensemble d'optimisations et nouvelles fonctionnalités.

Pour Angular, l'équipe de Google a décidé de partir sur un nouveau système de versioning : le semantic-versioning. Le terme Angular représente donc le Framework. Angular 2 représente la version majeure 2, Angular 4 représente la version majeure 4.

Quand on parle d'Angular, on parle donc de la version disponible depuis 2016, architecturé autour des composants, écrite en TypeScript, etc. Quand on parle d'AngularJS, il s'agit du Framework disponible depuis 2009, utilisant le pattern MVC et écrite en JavaScript.

Les nouveautés d'Angular 4

NgIf NgElse

L'une des nouveautés syntaxiques apportées par Angular 4 est la notion de else associée à la directive nglf. Jusqu'à maintenant, pour faire de l'affichage conditionnel, on utilisait la syntaxe suivante :

```
<div *ngIf="isLogged">
  The user is logged in.
</div>
<div *ngIf="!isLogged">
  The user is not logged.
</div>
```

Le premier nglf permet d'afficher un message si le booléen isLogged est à true. Le deuxième nglf inverse la condition pour afficher un message dans le cas contraire. La nouvelle syntaxe permet de se passer du deuxième nglf contenant la condition inversée :

```
<div *ngIf="isLogged; else login">
  The user is logged in.
</div>
<ng-template #login>The user is not logged.</ng-template>
```

Le nglf contient maintenant une instruction else définissant le nom du template à utiliser lorsque la condition n'est pas remplie. La base ng-template permet ensuite de définir le contenu du template associé au else.

Cette syntaxe permet de simplifier grandement les composants où il y avait beaucoup d'affichages conditionnels, notamment pour afficher des loaders lors des chargements, etc.

Une compilation AOT au top

Un des grands changements dans la nouvelle version d'Angular est une compilation Ahead of Time (AoT) améliorée.

Prenons le temps de rappeler ce qu'est l'AoT : en mode AoT, Angular compile les templates de l'application lors du processus de build en générant le code JavaScript nécessaire. Par opposition, en mode Just in Time (JiT), la compilation des templates se déroule au runtime, quand l'application se lance sur le client.

Ainsi, le flux d'une application Angular est le suivant en mode JiT :

- développement de l'application Angular en TypeScript ;
- compilation du TypeScript en Javascript (tsc) ;
- les bundles d'applications sont générés puis minifiés ;
- l'application peut être déployée.

A ce moment-là, en JiT, le JavaScript généré n'est pas encore compilé par le compilateur Angular, et c'est le client qui va devoir le faire, pour générer le code adéquat à chaque composant. Cela veut aussi dire que le client a besoin de récupérer les sources du compilateur Angular aussi pour effectuer cette compilation.

Grâce à la compilation AOT, le flux est légèrement différent :

- développement de l'application Angular En TypeScript ;

- l'application est compilée avec le compilateur Angular (@angular/compiler-cli) :
  - C'est un code TypeScript qui est généré par le compilateur Angular,
  - Le compilateur AOT permet de détecter les erreurs dans le template à la compilation, ce qui est pratique pour le développeur (et ce qui n'est pas le cas en JiT !),
  - Ensuite, ce code TypeScript est compilé en JavaScript,
- les bundles sont créés et minifiés.

Ainsi, il suffit au client de récupérer les bundles générés et l'application peut être rendue sans aucune compilation supplémentaire.
En termes de nouveauté, l'équipe d'Angular a mis en place un nouveau View Engine qui permet de réduire de manière très significative la taille des bundles finaux. Pour les applications étant au moins de taille moyenne, les gains sont de plus de 50 à 60%, et la différence est notable lorsque l'application s'agrandit. En ce qui concerne les petites applications, il est évident que, celles-ci ayant une base de code plus réduite, l'impact est moindre mais tout de même présent.

Un package dédié pour les animations

Dorénavant, les animations sont déplacées dans un package dédié : *@angular/platform-browser/animations*. Celles-ci se trouvaient dans le package principal *@angular/core* initialement. Cela veut dire que, dans un projet qui n'utilise pas les animations, le code relatif à celles-ci ne sera plus présent dans le bundle final.

La nouvelle pipe Titlecase

Parmi les nouveautés, une nouvelle pipe Titlecase a été intégrée et permet de mettre la première lettre de chaque mot en majuscule.
A l'utilisation :

```
<span> angular is the best </span>
```

Le texte sera : "Angular Is The Best".

Des Query String plus claires

Avec Angular 2, pour faire des requêtes http en Query String, il était nécessaire d'instancier un objet UrlSearchParams et d'y ajouter les paramètres.

```
let urlSearchParams = new URLSearchParams();
urlSearchParams.set("status", status.toString());
urlSearchParams.set("contratId", contratId.toString());
urlSearchParams.set("restaurantId", restaurantId.toString());
this.http.get(url, { search: urlSearchParams })
  .map(res => {
    let data = res.json();
    // Utiliser la donnée
  });
```

Dorénavant, le http.get prend directement un objet qui représente la donnée à envoyer en Query String :

```
this.http.get(url, {
  params: {
    status: status.toString(),
    contratId: contratId.toString(),
    restaurantId: restaurantId.toString()
  }
}).map(res => {
  let data = res.json();
  // Utiliser la donnée
});
```

Un service pour les meta tags

Un nouveau service est mis à disposition, le service Meta. Comme tous les autres services, il peut être injecté dans les composants et offre des méthodes pour pouvoir manipuler les meta tags de l'application. Le service Meta est présent dans le package *@angular/platform-browser*.

```
interface Meta {
  addTag(tag: MetaDefinition, forceCreation: boolean): HTMLMetaElement | null
  addTags(tags: MetaDefinition[], forceCreation: boolean): HTMLMetaElement[]
  getTag(attrSelector: string): HTMLMetaElement | null
  getTags(attrSelector: string): HTMLMetaElement[]
  updateTag(tag: MetaDefinition, selector?: string): HTMLMetaElement | null
  removeTag(attrSelector: string): void
  removeTagElement(meta: HTMLMetaElement): void
}
```

A l'utilisation, cela est assez intuitif :

```
constructor(private meta: Meta) {
  // Ajout d'un tag
  this.meta.addTag({ name: 'author', content: 'Programmez' });

  // Ajout de plusieurs tags à la fois
  this.meta.addTags([
    { name: 'author', content: 'Programmez' },
    { name: 'copyright', content: 'Programmez' }]);

  // Syntaxe d'un sélecteur
  let authorSelector = 'name="author"';

  // Récupération d'un tag (Retourne un HTMLMetaElement)
  let authorTag = this.meta.getTag(authorSelector);

  // Suppression d'un tag
  this.meta.removeTag(authorSelector);
}
```

Un validateur d'email

Un nouveau validateur a été ajouté aux Reactive Forms, il est ainsi possible d'utiliser le validateur email pour s'assurer de la validité d'un input de saisie d'email.

```
let emailControl = new FormControl("", Validators.email);
```

Une directive pour comparer les options d'un select

Il est possible de passer une méthode à la directive *compareWith* afin de comparer les options d'un select. Prenons l'exemple avec une entité Person qui possède les propriétés Id et Name.

```
interface Person {
  id: number;
  name: string;
}
```

On peut alors remplir le select de manière traditionnelle, tout en ajoutant la directive *compareWith* à qui on fournit la méthode de comparaison *compareById*.

```
<select [compareWith]="compareById" [(ngModel)]="selectPerson">
  <option *ngFor="let person of persons" [ngValue]="person">{{person.name}}</option>
</select>

compareById(personA:Person, personB:Person)
{
   return personA.id === personB.id;
}
```

Le guard CanDeactivate amélioré

Le guard CanDeactivate possède maintenant un paramètre optionnel supplémentaire *nextState*, cela permet de connaître la route sur laquelle la navigation veut être effectuée.

```
export interface CanDeactivate<T> {
   canDeactivate(component: T, currentRoute: ActivatedRouteSnapshot, currentState: Router
StateSnapshot, nextState?: RouterStateSnapshot): Observable<boolean> | Promise<boolean>
  | boolean;
}
```

Cela permet au développeur d'implémenter des use-cases spécifiques en fonction de la destination, au sein même du guard.

Comprendre la détection de changement

Un des paradigmes de base d'Angular, et même d'un framework qui s'occupe de faire du rendu, est la détection de changement. Chaque framework possède une approche différente de la détection de changement et les équipes d'Angular ont fait un effort considérable pour avoir une détection de changement la plus optimisée possible.

Cependant, lors du développement d'une application Angular, il est essentiel de comprendre le fonctionnement de cette détection de changement pour optimiser ses composants et ainsi avoir une application la plus réactive possible.

Ce qu'il faut savoir, c'est qu'une application est basée sur un ensemble de modèles. L'objectif est de présenter ces modèles sous une forme compréhensible pour l'utilisateur, dans notre cas : le DOM. Ainsi, Angular doit faire un rendu de ce DOM, et le mettre à jour lorsque le modèle change.

Cela veut donc dire qu'à chaque fois que le modèle change, l'application doit accéder au DOM, ce qui est très coûteux, il faut donc minimiser au maximum ces opérations.

Qu'est ce qui entraîne une détection de changement ?

Ce sont tout simplement les opérations asynchrones comme :
les évènements ;

```
<button (click)="doSomething()">Click sur moi</button>
```

les XHR ;

```
this.http.get('http://fakeUrl.com').subscribe(data => {
  let json = data.json();
});
```

les timers ;

```
setTimeout(() => {
  console.log("Angular > React ");
}, 1000);
```

Il faut alors déclencher une détection de changement lorsque ces opérations asynchrones sont lancées. Une mise à jour du DOM est alors requise.

Comment Angular est-il au courant qu'une opération asynchrone est levée ?

C'est grâce au concept des Zones, notamment grâce à la librairie Zone.js. Sans rentrer trop dans le détail, il est important de comprendre le fonctionnement de la call stack JavaScript. JavaScript étant mono thread, il ne peut faire plusieurs choses à la fois, on a alors une call stack tout à fait standard.

Or, dans le cas d'une opération asynchrone, le comportement est légèrement différent car ces opérations sont envoyées dans une queue du navigateur qui est la *event queue*. Lorsque la call stack normale est vide, les éléments présents dans la *event queue* sont alors dépilés.

C'est à ce moment-là que les Zones interviennent, en permettant de déclencher des actions lorsqu'un code commence à être exécuté au niveau de cet *event queue*.

En ce qui concerne Angular, le framework utilise sa propre Zone : NgZone, qui est une Zone « Monkey patché », en rajoutant du code avant et après l'exécution réelle du code asynchone.

```
zone.run(()=>{
  before();

  setTimeout(()=>{
    // l'exécution du code
  );
  after();
});
```

Très bien, et donc la détection en soi ?

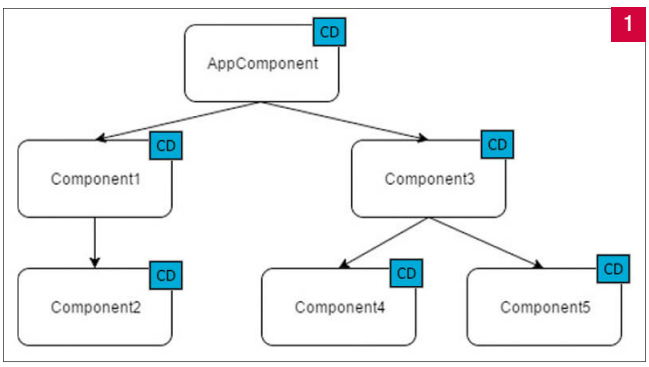
C'est une fonction nommée *tick* qui est responsable d'exécuter tous les détecteurs de changement. Chaque composant possède son propre détecteur de changement ; c'est une très bonne nouvelle car il est possible de gérer la détection de changement de manière unitaire sur chaque composant.

Or, une application Angular est un arbre de composants, cela veut dire que l'on a un arbre de détecteurs de changements aussi. [1]

Cet arbre de détecteurs de changements est appelé graphe de détection, c'est un arbre dirigé dont le flux de données est unidirectionnel, en partant du haut. Cela veut dire que le chemin est prévisible et qu'on peut alors parcourir l'arbre plus rapidement.

Niveau performance, la détection d'Angular est conçue pour être la plus rapide possible, le code qui est généré par Angular est adapté à la machine virtuelle JavaScript.

Cependant, cela n'empêche pas que le développeur puisse optimiser cer-



taines choses, notamment en évitant de solliciter les détecteurs de changement lorsque ce n'est pas nécessaire.

Immuabilité + OnPush = <3

Ce qu'il faut comprendre, c'est qu'Angular parcourt l'arbre de détecteur de changement en entier à chaque fois. Cela est dû au fait qu'Angular doit vérifier tous les composants, dont les objets pouvant avoir des propriétés modifiées. Pourquoi ? Car le framework suppose que par défaut, les objets sont variables, c'est-à-dire non-immuables.

Pour rappel, un objet immuable est un objet qui ne peut pas changer, et les objets JavaScript ne sont pas immuables, sauf pour les primitives car ce sont des types valeurs et non des types références.

En utilisant des librairies JavaScript, il est possible de créer des objets immuables. Ainsi, pour changer la propriété d'un objet, un nouvel objet est créé, avec une nouvelle référence. Nous avons alors une politique de modification unique : une modification == une nouvelle référence.

```
let personA = {
  firstName: "Daniel",
  lastName: "Djordjevic"
};

let personB = Object.assign({}, personA, { firstName: "William" });
// personA === personB -> Renvoie false car ce n'est pas la même référence
```

Dans l'exemple ci-dessus, avec Object.assign, on crée une première personne *personA*. Puis, avec *Object.assign*, on crée un nouvel objet, dans lequel on injecte les données de *personA*, puis on modifie la propriété *firstName*.

L'objet *personB* est alors un nouvel objet, avec une référence différente de l'objet *personA*.

Avec ce concept d'immuabilité et un composant qui ne dépend que de ses inputs, il est alors possible d'utiliser la stratégie de détection de changement alternative d'Angular : *OnPush*.

Prenons l'exemple avec un composant PersonComponent :

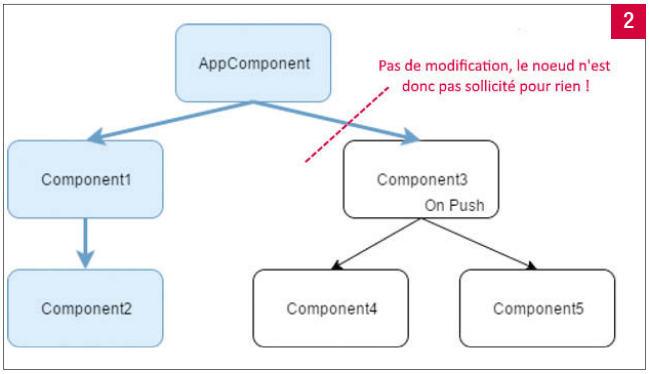
```
@Component({
  selector: 'person',
  template: '<pre> {{person | json}}</pre>',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class PersonComponent {
  @Input() person: Person;
}
```

Le composant affiche le contenu de la propriété *person* grâce à la pipe *json* dans une balise *pre*. Cette propriété *person* est un input, et la détection de changement est en mode *OnPush*.

Avec le concept d'immuabilité, le détecteur de changement ne sera sollicité que lorsque l'objet person change de référence, et c'est une très bonne nouvelle. De plus, comme nous sommes dans un arbre unidirectionnel, si on sait bloquer un détecteur de changement à un niveau, on évite de solliciter les détecteurs de changement sous-jacents. [2]

Typescript 2.2

Pour cette nouvelle version d'Angular, les équipes de Google ont travaillé à améliorer la compatibilité avec Typescript vers ses dernières versions. La compatibilité annoncée est donc à la version 2.2 de Typescript. En revanche, la CLI ne se prive pas pour utiliser des versions encore plus



récentes sans que cela ne pose de problème.

Angular 4 peut donc profiter des dernières nouveautés de TypeScript et notamment celles qui suivent.

Null-Check sur les opérateurs

Cette fonctionnalité va vérifier à la compilation si des opérandes peuvent être null et si cela peut poser problème. Si c'est le cas, une erreur de compilation est levée.

Les opérations concernées sont les suivantes :

- + lors qu'une des opérandes est de type any ou string ;
- -, *, **, /, %, <<, >>, >>>, &, |, et ^ lorsqu'une des opérandes est nullable ;
- <, >, <=, >=, et in lorsqu'une des opérandes est nullable ;
- instanceof lorsque l'opérande de droite est nullable ;
- +, -, ~, ++, et – lorsque l'opérande unitaire est nullable.

Par exemple, le code suivant provoque une erreur de compilation :

```
function lengthValidator(input: string, min: number, max?: number) {
  return input.length >= min
    && input.length <= max;
}
```

En effet, ici le paramètre max peut-être null alors qu'il est utilisé dans une opération <=.

Le type object

Le nouveau type object qui s'intégre à la version 2.2 de Typescript représente tous les types qui ne sont pas primitifs. Si une méthode attend un type object, elle ne pourra donc pas être utilisée avec les types suivants :

- Boolean ;
- Number ;
- String ;
- Symbol ;
- Null ;
- Undefined.

Amélioration de la signature d'index

Avec TypeScript 2.2, il est maintenant possible d'accéder aux propriétés déclarées en signature d'index comme à une propriété traditionnelle :

```
interface Dictionary<T> {
  [key: string]: T;
}

const numberDictonary: Dictionary<number> = {};
//erreur avant typescript 2.2
```

```
numberDictonary.trois = 3;
//l'ancienne syntaxe est toujours valide
numberDictonary['trois'] = 3;
```

Amélioration de support des mixins

Avec cette nouvelle version de TypeScript, il est maintenant possible de créer des dérivés de classe très facilement. Pour cela, il faut :

- Déclarer un constructeur ;
- Déclarer une classe qui étend le constructeur ;
- Ajouter des membres à cette classe ;
- Retourner la classe nouvellement créée.

Par exemple, pour ajouter une date à une classe de manière dynamique, il est possible de procéder de la sorte :

```
/* N'importe quel type disposant d'un constructeur. */
export type Constructable = new (...args: any[]) => {};

export function Timestamped<BC extends Constructable>(Base: BC) {
  return class extends Base {
    timestamp = new Date();
  };
}

class Point {
  x: number;
  y: number;
  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }
}

const TimestampedPoint = Timestamped(Point);

const p = new TimestampedPoint(10, 10);
const sum = p.x + p.y;
p.timestamp.getMilliseconds();
```

Faire le rendu coté serveur

Une des grandes nouveautés de la version 4 du Framework de Google est l'intégration de la gestion du rendu coté serveur. Le rendu coté serveur permet de retourner un HTML pré-loadé par le serveur au lieu de fournir à l'application le fichier index.html tel quel. Ceci permet d'afficher des données avant même que l'application Angular ne soit bootée dans le navigateur.

En effet, auparavant ce concept ne pouvait se réaliser qu'en utilisant la librairie tierce Angular Universal. C'est d'ailleurs de ce repository GitHub que le merge s'est effectué vers Angular ; la team de Google n'est pas partie d'une feuille de blanche et a préféré ne pas réinventer la roue.

Intérêt

Le rendu coté serveur d'une application front permet d'apporter deux améliorations notables.

La première concerne le référencement. En effet les crawlers de moteurs de recherche, tels que celui de Google par exemple, n'exécutent pas toujours le Javascript. Ils se retrouvent alors avec une application non bootstrappée et sont donc incapables de traiter son contenu. Avec le rendu côté serveur, c'est une page HTML déjà remplie qui est retournée par le serveur, les crawlers peuvent donc y lire le contenu sans problème.

La seconde amélioration concerne l'expérience utilisateur. En effet avant qu'une page d'application front s'affiche, il faut que le navigateur télécharge la page HTML, télécharge les dépendances CSS et JS de celle-ci, interprète le Javascript, et, enfin, que le framework bootstrappe l'application. En utilisant le rendu coté serveur, le navigateur télécharge directement une page HTML pré-rendue qu'il peut afficher convenablement dès que le CSS est chargé. Il faut en revanche garder en tête que l'application ne sera interactive qu'une fois qu'elle aura été bootstrappée.

Voici un exemple concret de la différence de performance en utilisant une application qui effectue un simple listing. Pour ce test, j'ai utilisé les outils de développement Chrome afin de simuler un réseau 4G et j'ai désactivé le cache du navigateur. J'ai également activé l'enregistrement de frames lors du chargement de la page afin de pouvoir évaluer la différence, en termes de vitesse d'affichage de l'application, avec et sans rendu coté serveur : [3]

Dans le cas d'un chargement de l'application sans rendu coté serveur, le navigateur affiche « Loading » (Le message du template de base d'une application Angular) tout le temps du chargement des fichiers Javascript. Il faut alors plus de 7 secondes à l'application avant de s'afficher. En revanche lorsque l'on utilise le rendu coté serveur, il suffit que le fichier HTML soit chargé pour afficher des données. La page s'affiche au bout seulement 200 millisecondes. Il faut en revanche attendre que tout le Javascript soit chargé, soit plus de 6 secondes, avant que l'application ne soit interactive.

Principe

Le principe du rendu coté serveur est d'avoir un serveur node.js (ou autre) qui utilise la même application Angular que le client.

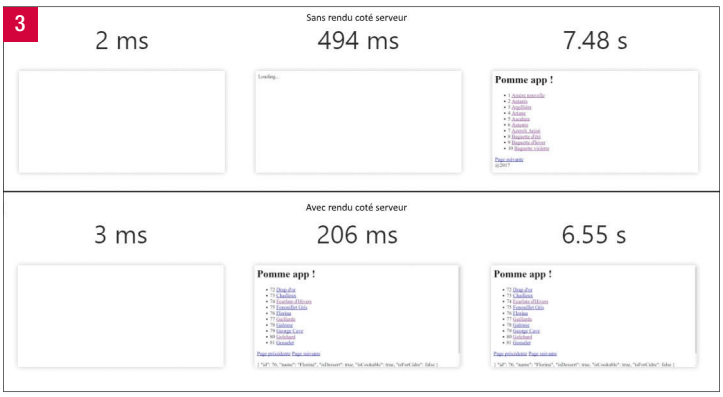
Le serveur va se servir de cette application pour générer le HTML en fonction des arguments passés dans la requête HTTP.

Le module Angular que doit exécuter le serveur ne peut en revanche pas être le même que celui utilisé par le client. En effet le module dont se sert le client fait référence à des modules d'Angular qui nécessitent l'exécution au sein d'un navigateur (comme par exemple le module `BrowserModule`).

Voici comment communiquent ces différents éléments : [4]

Dans le schéma, lorsque le navigateur effectue une requête HTTP (1), le serveur va utiliser son module spécifique dans l'application Angular pour générer le HTML (2). Il va ensuite retourner cet HTML dans la réponse HTTP vers le navigateur (3). Une fois que le navigateur a récupéré cet HTML, il va l'afficher puis charger l'application Angular en servant du module principal (4).

3



Pour mettre ce mécanisme en plus il faut donc commencer par créer un module pour le serveur qui n'utilise pas les modules d'Angular nécessitant l'exécution au sein d'un navigateur. Ensuite il faut créer le serveur node.js se servant de ce module. Enfin, il faut créer la configuration de compilation de ce module.

Mise en place

Module serveur

Afin de mettre en place le rendu coté serveur, il faut dans un premier temps créer un module qui exécutera l'application en dehors du navigateur. Pour créer ce module, il faut ajouter les dépendances suivantes au projet :

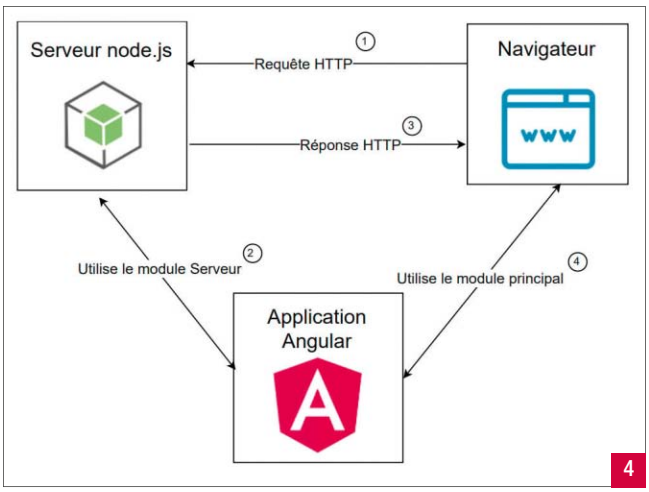
```
npm install --save-dev @angular/animations
npm install --save-dev @angular/platform-server
```

Une fois ces dépendances installées, il ne reste plus qu'à créer un module de la manière suivante :

```
import { NgModule } from '@angular/core';
import { ServerModule } from '@angular/platform-server';
import { AppComponent } from './app.component';
import { AppModule } from './app.module';
@NgModule({
 imports: [
      ServerModule,
      AppModule
 ],
 bootstrap: [
      AppComponent
 ]
})
export class AppServerModule {}
```

Ce module pour le serveur doit importer le ServerModule depuis @angular/platform-server ainsi que le module principal de l'application. Il faut également ajouter le composant principal de l'application dans la propriété bootstrap.
Il faut également modifier le module client de l'application pour que celui-ci puisse gérer la transition entre la page HTML statique de l'application et l'application dynamique qu'il doit bootstrapper.

```
import ...

@NgModule({
 declarations: [...],
 imports: [
  BrowserModule.withServerTransition({appId : "pomme-app"}),
  ...
 ],
 providers: [...],
 bootstrap: [AppComponent]
})
export class AppModule { }
```



4

L'appId à passer en paramètre sert simplement d'identifiant de l'application. On peut utiliser n'importe quelle valeur

Compilation AOT

Afin de fonctionner, le module serveur a besoin des factory de modules obtenus par la compilation AOT (Ahead Of Time). Pour cela, il faut créer un fichier tsconfig.server.json contenant la configuration suivante :

```
{
 "compileOnSave": false,
 "compilerOptions": {
  "outDir": "./dist/out-tsc",
  "baseUrl": "src",
  "sourceMap": true,
  "declaration": false,
  "moduleResolution": "node",
  "emitDecoratorMetadata": true,
  "experimentalDecorators": true,
  "target": "es5",
  "typeRoots": [
   "node_modules/@types"
  ],
  "lib": [
   "es2016",
   "dom"
  ]
 },
 "files": [
  "src/app/app.server.module.ts"
 ],
 "angularCompilerOptions": {
  "genDir": "src/aot",
  "entryModule": "./src/app/app.server.module#AppServerModule"
 }
}
```

Il s'agit d'une copie du fichier tsonfig.json se situant à la racine du projet, cependant le nœud "files" pointe vers le fichier du module server créé précédemment et un nœud angularCompilerOptions a été ajouté. Celui-ci précise le dossier où doivent se situer les fichiers générés ainsi que le chemin et la classe du module serveur.

Création du serveur Node.JS

Ici commence la partie moins triviale. Si vous souhaitez mettre en place ce concept sans en comprendre le fonctionnement détaillé, le fichier du serveur complet est disponible en fin de cette partie.
Afin de créer le serveur node.js, on a besoin d'ajouter quelques dépen-

dances au projet :

```
npm install webpack @ngtools/webpack raw-loader express @types/express --save-dev
```

Il faut ensuite créer le serveur Web. Pour cela, il suffit d'ajouter un fichier main.server.ts à la racine du projet. Dans ce fichier, la première chose à faire est d'activer le mode de production d'Angular :

```
import { enableProdMode } from '@angular/core';

enableProdMode();
```

Ensuite il faut récupérer la référence à Express et créer une nouvelle application. Express est le serveur Web node.JS dont nous allons nous servir.

```
import * as express from 'express';

const app = express();
```

Afin que ce même serveur puisse également servir à retourner les fichiers nécessaires à l'exécution de l'application Angular, il faut ajouter un middleware de fichier statique :

```
app.use(express.static('.'));
```

Enfin on peut lancer le serveur Express sur un port quelconque.

```
app.listen(8000, () => {
  console.log('listening...');
});
```

A ce stade, nous avons simplement créé un serveur de fichier statique. Il faut ensuite créer un moteur de modèles Express. Ce genre de moteur prend une vue (html dans ce cas) et des paramètres en entrée et fournit une vue html interprétée en sortie.
Pour créer un moteur, il faut appeler la méthode `engine` sur l'application express. Cette méthode prend en paramètre le type des fichiers d'entrées (ici html) et une fonction devant effectuer la transformation.

```
app.engine('html', ngEngine);
```

Reste à définir le moteur en lui-même. Pour cela il suffit d'écrire une méthode prenant 3 paramètres. Le premier correspond au chemin vers le fichier html demandé. Le second paramètre est un objet qui est construit lors des appels au moteur de modèle. C'est à nous de remplir (ou non) cet objet. Le dernier paramètre est un callback qui prend lui-même deux paramètres. Il faut utiliser le 1er paramètre lorsqu'une erreur s'est produite, et le second lorsque tout s'est bien passé.
Il faut utiliser le premier paramètre du callback en cas d'erreur et le second en cas de succès contenant la chaîne de caractères que le serveur doit retourner.

```
const ngEngine = (filePath, options, callback) => {
  const file = fs.readFileSync(filePath).toString();
  callback(null, file);
};
```

Pour le moment ce moteur de vue ne fait que charger la page html demandée avant de la retourner. Afin d'interpréter la page HTML avec notre application Angular, il faut utiliser la méthode renderModuleFactory située dans @angular/platform-server. Cette méthode prend deux paramètres. Le premier est la classe de factory associée au module serveur de l'application. Le second est un objet de type PlatformOption. Celui-ci contient une propriété document qu'il faut remplir en utilisant le contenu du fichier HTML sur lequel l'application doit bootstrapper, et une propriété url qui doit contenir l'url appelée.

```
import { renderModuleFactory } from '@angular/platform-server';
import { AppServerModuleNgFactory } from './aot/src/app/app.server.module.ngfactory';
import * as fs from 'fs';

const ngEngine = (filePath, options, callback) => {
  const file = fs.readFileSync(filePath).toString();
  renderModuleFactory(AppServerModuleNgFactory, {
    document: file,
    url: options.req.url
  })
};
```

La méthode renderModuleFactory retourne une promise. Le résultat de cette promise contient la chaîne de caractères de l'html rendu. Il ne reste donc plus qu'à appeler le callback dans le résultat de cette promise. Voici donc le code complet concernant le moteur de modèle :

```
const ngEngine = (filePath, options, callback) => {
  const file = fs.readFileSync(filePath).toString();
  renderModuleFactory(AppServerModuleNgFactory, {
    document: file,
    url: options.req.url
  })
  .then(string => {
    callback(null, string);
  });
};

app.engine('html', ngEngine);
```

Pour que ce moteur soit utilisé, il faut le préciser à Express en utilisant le méthode set sur l'application avec en paramètre 'view engine'. Le deuxième paramètre de cette méthode doit être le type de fichier géré par le moteur (ici html). Il faut également lui spécifier le répertoire contenant les vues (ici c'est la racine du serveur) en utilisant cette même méthode avec le paramètre 'views'.

```
app.set('view engine', 'html');
app.set('views', '.');
```

Maintenant que le moteur de modèles est en place, il faut l'appeler sur toutes les urls qui peuvent être demandées au serveur. Il faut donc utiliser la méthode « get » de l'application Express. Cette méthode prend deux paramètres en entrée. Le premier correspond à l'url, ou à un pattern d'urls, à laquelle l'application doit répondre. Le second est la fonction qui effectue la création de la réponse HTTP. Cette fonction prend en paramètre la requête et la réponse HTTP. C'est sur cet objet response qu'un l'appel à la méthode render est effectué afin de faire appel au moteur de modèles.

```
app.get('*', (request, response) => {
  response.render('index', { req: request });
});
```

Le deuxième paramètre passé à la méthode render correspond à l'objet option qui est récupéré dans le moteur de modèles. On lui passe donc ici la requête afin que le moteur puisse en déduire l'url appelée.

Le fichier final doit donc ressembler à cela :

```
import { renderModuleFactory } from '@angular/platform-server';
import { enableProdMode } from '@angular/core';
import { AppServerModuleNgFactory } from './aot/src/app/app.server.module.ngfactory';
import * as express from 'express';
import * as fs from 'fs';

enableProdMode();

const app = express();

const ngEngine = (filePath, options, callback) => {
 const file = fs.readFileSync(filePath).toString();
 renderModuleFactory(AppServerModuleNgFactory, {
  document: file,
  url: options.req.url
 })
 .then(string => {
  callback(null, string);
 });
};

app.engine('html', ngEngine);

app.set('view engine', 'html');
app.set('views', '.');

app.use(express.static('.'));

app.get('*', (request, response) => {
 response.render('index', { req: request });
});

app.listen(8000, () => {
 console.log('listening...');
});
```

Compilation

Afin de compiler ce serveur, il faut utiliser webpack. Il faut donc définir un fichier webpack.config.ts permettant d'effectuer la compilation du serveur. Il s'agit ici d'un fichier de configuration assez traditionnel mis à part le fait qu'il faille ajouter un plugin pour la compilation AOT de l'application et qu'il faille utiliser le loader @ngtools/webpack sur les fichiers de type Typescript :

```
const ngtools = require('@ngtools/webpack');
const config = {
 entry: {
  main: './src/main.server.ts'
 },
 resolve: {
  extensions: ['.ts', '.js']
 },
 target: 'node',
 output: {
  path: __dirname + '/dist',
  filename: 'main.server.js'
 },
 plugins: [
  new ngtools.AotPlugin({
   tsConfigPath: './tsconfig.server.json',
  })
 ],
 module: {
  rules: [
   { test: /\.css$/, loader: 'raw-loader' },
   { test: /\.html$/, loader: 'raw-loader' },
   { test: /\.ts$/, loader: '@ngtools/webpack' }
  ]
 }
}
module.exports = config ;
```

Pour aller plus loin :

Si le sujet vous intéresse et que vous voulez approfondir, n'hésitez pas à lire notre livre : *Angular, Développez vos applications web avec le framework JavaScript de Google.*(http://bit.ly/2umr7Nv)

•

LesFurets.com : “mettre chaque jour du code en production !”


François Tonic


Dimitri Baeli

Le site LesFurets.com est un comparateur d'assurances. Chaque jour, le site doit supporter des milliers de requêtes pour produire des milliers de devis. La livraison continue, le déploiement continu du code ne sont pas des concepts abstraits pour les 25 développeurs. Aujourd'hui, ce sont plus de 260 releases qui sont livrées et déployées chaque année ! Cela est rendu possible grâce à deux approches très complémentaires. D'un côté Kanban pour organiser un développement en flux dépassant le cadre classique des Sprints agiles. De l'autre, une plateforme de déploiement continu : Git & Jenkins adaptée aux besoins réels des équipes. Explications détaillée de l'utilisation de Git avec Dimitri Baeli, le directeur technique (CTO).

Pour les besoins internes des équipes de développement, un projet a été initié sur la base Git : Git Octopus. “Nous n'utilisons pas que des outils out of the box. On veut aussi s'approprier les outils, organiser le développement au quotidien.” introduit Dimitri.

Aucun développeur n'est bloqué par un autre !

L'idée de départ est très simple : pouvoir développer plusieurs fonctionnalités en même temps, et pouvoir envoyer en production n'importe quelle feature dès qu'elle est prête. L'idée fondatrice du CTO est très simple : “je veux publier chaque jour tout ce qui est prêt, sans souffrir d'une planification minutieuse des jours à l'avance et éternellement à remanier !” Travailler sur les sujets les plus importants, et publier ce qui est prêt, sans s'épuiser à planifier : une philosophie défendue par l'approche Lean Kanban, et matérialisée grâce à l'outillage Git/Jenkins que nous allons vous décrire.

La philosophie appliquée est qu'aucun développeur ne puisse bloquer les 24 autres développeurs. Les codes sont séparés pour que tout fonctionne. C'est le concept de continuous merge et de feature branching.

La personnalisation de Git a été nécessaire pour “forcer” le fonctionnement de l'outil et permettre une gestion très fine des branches, avec la notion de feature branching (voir encadré). Et si des conflits arrivent durant les merges, le problème est immédiatement identifié et peut être traité de différentes manières souvent très simples.

Ce projet vient de l'extrême agilité dans les équipes. “Notre idée est de livrer tous les jours” précise notre CTO. La méthode Kanban a été l'inspiration. Kanban a été édictée en 2010 et repose sur le principe de la limitation du nombre de travaux à faire. L'objectif est d'éviter le gaspillage des ressources et du temps, et d'assurer une amélioration continue des processus. Dans une approche Kanban, le travail est divisé en tâches indépendantes.

Il s'agit donc ici de garder le plus longtemps possible les développements indépendants les uns des autres. “Publier quotidiennement sans faire de branche pousse à publier toutes les étapes intermédiaires, ce qui ne correspond pas à une fonctionnalité complète !” résume Dimitri. “Bref, on isole les développements de chaque développeur, toujours selon l'esprit du feature branching. C'est un défi pour le travail en équipe. Mais pour pouvoir publier du code chaque jour, il faut faire des choix… Il faut voir notre base de code comme un gros dictionnaire. On fait une petite modification dans une définition, et on veut la publier mais j'ai 25 personnes sur la même base. Mais comme nous n'avons pas fait de micro-projets ou de micro-services, cette indépendance de développeur est cruciale.

Ce besoin d'indépendance des tâches en livraison nous a poussé à tester le feature branching qui est curieusement l'inverse de ce que l'on recommande, mais nous savons pourquoi nous le faisons et que grâce à l'intégration continue réinventée nous sommes bien protégés” martèle Dimitri. LesFurets.com voulaient faire du continuous delivery avec trunck-based mais finalement, le feature branching avec Git est devenu la pierre angulaire de l'organisation des développements.

Le feature branching assure donc un développement isolé entre les développeurs. Et quand une branche est prête, on peut livrer, sans se soucier du reste. Si vous êtes un habitué des usines logicielles , vous devinez ce que l'on risque de perdre avec cette approche : l'intégration continue et la résolution des conflits entre branches [1].

Mais l'idée n'est pas de perdre l'intégration continue qui est tout de même le socle technique indispensable … Il faut donc concilier deux approches qui semblent peu compatibles. La solution trouvée est le “continuous merge”.

FEATURE BRANCHING : des branches et de l'intégration continue ? Si, si, on peut le faire

Tout part du master (au niveau de la branche de code), puis on crée une branche de développement, par exemple features/f1, puis une 2e... On applique ensuite un merge automatique de toutes les branches ensemble pour reconstruire le code à tester. L'intégration continue s'applique sur le merge automatique, en laissant chaque branche indépendante.

Donc le code utilisant une approche feature branching fait que chaque feature du projet est une branche dédiée. Et donc, la release sera la fusion du master et des branches features de celui-ci.

[1] Oh you'r using feature branching ? Too bad, you're not doing continuous integration.

25 développeurs, 500 000 lignes de codes en production

Avant de généraliser Kanban, LesFurets.com étaient sur du Scrum classique avec des itérations et des livraisons tous les 15 à 30 jours. Kanban est très différent de Scrum pour l'aspect planification et livraison, on passe d'un modèle de lot à un modèle du flux (du solide au liquide).

Il a fallu revoir les méthodes de développement au quotidien. La méthodologie utilisée est donc assez différente de ce dont on vous parle tous les mois : agilité hors Scrum, du réel DevOps avec des développeurs qui s'occupent que de leur fonctionnalité, du développemment jusqu'à la mise en production.

"Une de nos philosophies est de livrer le code qui est prêt. Au quotidien, chaque développeur doit pouvoir travailler dans son coin, mais la segmentation fine des fonctionnalités évite que les développeurs n'empiètent sur le travail d'un collègue".

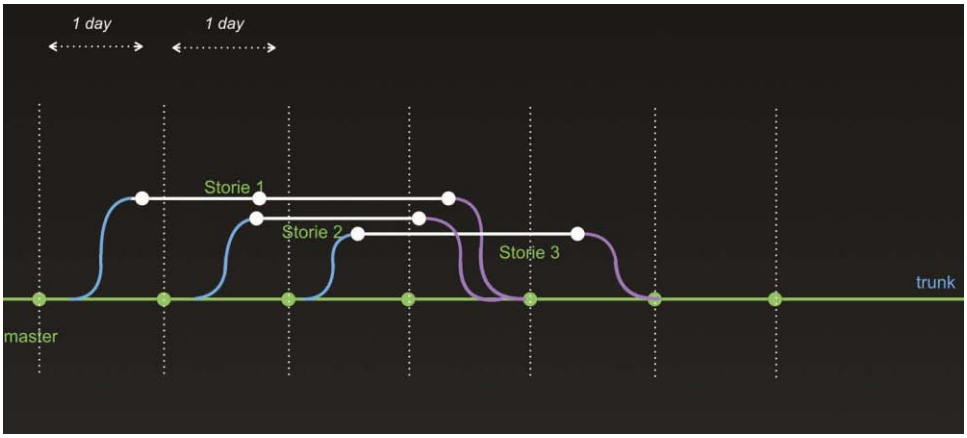
L'objectif est de fournir cette flexibilité extrême et de mettre en production ce qui est prêt tout en utilisant les principes du feature branching (permettant l'indépendant des développements entre eux) et l'intégration continue grâce au continuous merge. Cela signifie que le merge va merger les branches features avec la branche master. Si un développeur fait un commit sur la sous branche feature/f1, la fusion est automatique.

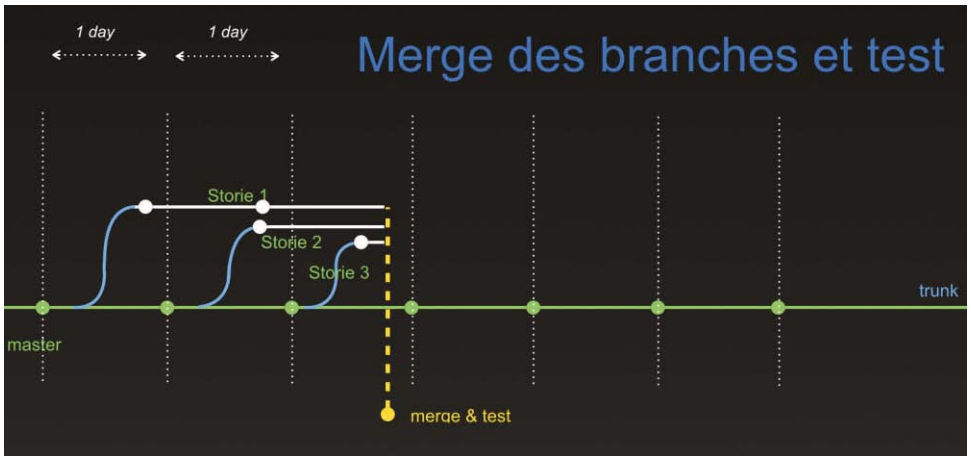
Mais faire une fusion de plusieurs branches à partir d'un pattern n'existe pas dans Git, d'où le projet Git-Octopus.

"Le Continuous Delivery n'intéresse plus que l'organisation agile"

Dimitri, CTO.

Avant le développement du projet Git-Octopus, LesFurets.com utilisaient le classique SVN dont la gestion des branches était très pénible, car aucun historique local n'était disponible. Puis les équipes sont passées à Git, notamment par son modèle de gestion du code distribué pour simplifier le travail local des développeurs, et les aider à travailler plus simplement. "L'objectif était clair : passer d'une publication mensuelle à une publication journalière" reprécise Dimitri. Ce changement assez radical ressemble à une industrialisation du process de patch. En effet, quand un bug apparaît en production il faut pouvoir le corriger et redéployer le code très ra-





GÉRER LES CONFLITS : UNE PRIORITÉ D'OCTOPUS

Quand on ajoute des dépendances entre deux branches, il y a des risques de conflits et donc la résolution peut être caduque ; et surtout on casse la logique même de feature branching. Il faut définir et utiliser des conventions de code communes à tous les développeurs, avoir un code modulaire avec de l'évolution par abstractions. Ainsi, on réduit les risques. Si un conflit entre 2 branches est détecté, on a tout un éventail de choix pour résoudre ce conflit : on peut en général éviter ce conflit, on peut mettre de côté l'une des deux, ou faire un rebase de l'une sur l'autre pour résoudre le conflit même si c'est déconseillé. Une des règles d'or de LesFurets.com : éviter les conflits plutôt que les résoudre.

"Avec ce mode de fonctionnement nous avons compris que résoudre un conflit entre deux branches au sein d'un même code revient à fusionner les dates de mise en production des deux fonctionnalités concernées. C'est donc une difficulté dans un contexte de continuous delivery" explique Dimitri.

Jenkins + Git + Git-octopus : le bonheur est dans le merge

L'usine logicielle est entièrement basée sur Jenkins. Une séquence de merge y a été rajoutée en toute première étape. Si un problème survient : l'intégration continue automatisée se casse, et l'équipe regarde la cause et la branche causant cette casse. Le développeur "en cause" est informé quelles branches sont en conflits. "Dans 99 % des cas, le conflit est évitable, et quand ça arrive, on modifie une seule branche. Il faut absolument éviter les conflits de branches. En 3 ans, nous avons passé quasiment 4000 branches dans notre système !"

Le site de Git Octopus : https://github.com/lesfurets/git-octopus

pidement ; d'où avoir un système de patch solide et surtout adapté à une publication en continu.
La question de la pression supplémentaire sur les développeurs est naturelle mais pour notre CTO, il s'agit d'un faux problème. "Je dirais non. Le moins de cette organisation est l'isolement du développeur car il n'a plus besoin de parler (à ses collègues), il code et il publie. Mais dans la pratique, nous avons supprimé les points de contention, les goulets d'étranglement des processus de développement et de déploiement. Surtout, nous supprimons la pression de la livraison notamment avec la fin du blocage potentiel de l'équipe par un seul développement qui serait en retard."
Un effet de bord est apparu avec cette organisation : moins de moments de décompression. Le continuous delivery peut être comparé à un marathon : tenir sur une longue distance, avec un rythme donné. Si le développeur va trop vite, il s'épuise. S'il va trop lentement, il sort du planning de développement.

Parler fonction et non plus contrainte technique

Cette organisation a aussi une incidence pour le développeur. Quand il termine son développement, il prend la fonction suivante à coder. Il y a donc une variété dans son travail même s'il peut tomber sur un développement qu'il n'aime pas faire. Cette variété sous-entend aussi que parfois, il ne va pas connaître une technique, une technologie, donc il va apprendre, se former. Le développeur devient plus polyvalent et sort de sa zone de confort.

Git est mon arme

Un des standards du développement "moderne" est la revue de code. Le code doit être compréhensible par un développeur externe qui doit l'approuver ou non et pouvoir modifier et maintenir ce code si besoin.
Le script Git-Octopus a été créé par un développeur qui n'est plus chez LesFurets.com : Arnaud Pflieger. Le projet est open source et s'intègre totalement à Git pour qu'il fasse ce que l'on souhaite qu'il fasse. Octopus existe pour répondre à la nécessité de mettre en production du code tous les jours. Mais avant la création d'Octopus, toute les pratiques avaient été passées en revue : Git flow, Github flow, la gestion des branches de développements et de production, etc. Mais rien n'a réellement convaincu Dimitri.
"Un jour, une idée est venue : mettre tout le code dans les branches et on fait un merge de toutes ces branches pour tester l'ensemble. On réinvente l'intégration continue à notre sauce pour que tout soit automatisé. Le merge est trop dangereux pour qu'il soit fait à la main par quelqu'un. J'ai été convaincu par Arnaud et il a testé et monté le script. Bien entendu, si un problème survient, nous sommes prévenus immédiatement, avant la mise en production et on sait quelle branche est concernée par le problème" poursuit Dimitri. Octopus était né.

2012
Deux releases en sprint scrum mensuel

2013
45 releases en passant en mode hebdomadaire

2014
208 releases en utilisant l'approche livraison quotidienne (kanban) et continuous merge

2015
225 releases

2016
260 releases

La base du script Git-Octopus est assez simple : utilisation de la stratégie de merge de plusieurs branches nommée "octopus". Et surtout, cela permet de :

- Garder l'intégration continue ;
- Livrer ce qui est prêt via le feature branching ;
- Avoir un investissement minimum ;
- Faciliter le passage à l'échelle
- Construire facilement les environnements de développement avec l'ensemble des fonctions et des outils nécessaires pour le code, la validation, la préproduction. •

Le magazine culturel des mathématiques, pour mieux comprendre le monde

Politique, économie, finance, jeux, musique, littérature, arts plastiques, architecture, informatique, physique, biologie, géographie : les mathématiques sont partout !!!
Le magazine *Tangente* et ses hors séries vous aident à redécouvrir notre quotidien.
Chaque année, paraissent 6 numéros normaux, 4 hors séries « kiosque » et 4 ouvrages de la Bibliothèque Tangente, versions augmentées des hors séries.

Les hors séries « kiosque »

4 fois par an, paraît un hors série (format « kiosque »).
Ces numéros d'au moins 56 pages explorent alternativement un grand dossier de savoir mathématique et un grand dossier culturel qui fait le lien avec une autre discipline, un art, un sujet d'actualité…
Ces hors séries sont inclus dans l'abonnement PLUS.

Les trésors de la bibliothèque Tangente pour les abonnés SUPERPLUS et SOUTIEN

Pour les amateurs les plus curieux, les articles des hors séries de *Tangente* sont repris et complétés dans les livres de la Bibliothèque Tangente (voir en pages 4 à 7 de ce catalogue), magnifiques ouvrages d'au moins 160 pages, richement illustrés, disponibles en librairie, sur la librairie du site Infinimath **www.infinimath.com/librairie** ou à prix de souscription avantageux avec l'abonnement SUPERPLUS ou l'abonnement de SOUTIEN.

La version numérique de *Tangente*

Un véritable site interactif augmenté et pas un simple pdf !
Il est maintenant possible de consulter en ligne sans limitation de durée les numéros et hors séries de *Tangente* sur le site **tangente-mag.com** pour tous ceux qui souscrivent à n'importe quelle formule d'abonnement, qu'elle soit « papier » ou numérique.
L'abonnement permet aussi d'accéder gratuitement aux problèmes du « Monde » sur **www.affairedelogique.com**

Abonnez-vous à *Tangente* !

- **en ligne sur www.infinimath.com/librairie**
- **par courrier à Éditions POLE - 2 rue La Prée - 27170 COMBON**

Téléchargez le catalogue POLE

sur http://infinimath.com/editionspole/catalogue_POLE_2017_I.pdf
Vous y trouverez certainement votre bonheur !

Cobol, mainframe : les légendes ne meurent jamais

François Tonic

Depuis plus de 40 ans, Cobol et mainframe règnent sur de nombreuses administrations et entreprises. Des applications critiques tournent uniquement dessus. Quasiment les ¾ des transactions commerciales passent par une application en production sur un mainframe. Et plus de 220 milliards de codes Cobol tournent dans le monde.

Ces chiffres montrent le poids considérable de Cobol et du mainframe dans l'économie et l'administration publique. Les besoins en maintenance sont énormes car pour continuer à faire tourner ces codes, il faut entretenir et maintenir les machines et les logiciels. Cependant, les nouveaux codes Cobol sont moins nombreux, au profit de technologies plus modernes, et le mainframe, en tant que machine, n'est plus une plateforme matérielle que l'on achète. On maintient.

Le mainframe subit le sort du marché serveur : de fortes baisses. Ainsi, selon une étude Gartner sur ce marché, le constat est sévère pour le début de l'année : baisse des livraisons, baisse du chiffre d'affaires global. IBM subit une très forte baisse (-34 %), en incluant les serveurs et les mainframes.

Même si le conservatisme existe, des entreprises n'hésitent pas à migrer vers des plateformes plus modernes. Plusieurs ont annoncé cette migration : Bergère de France, CNAF, Experian.

La perte de compétences peut aussi paralyser les initiatives de migration ou de réécriture. Quand le système fonctionne, on n'y touche pas, d'autant moins, si plus personne, ou presque, ne le comprend.

Cobol : toujours actif

Vieux, dépassé, inutile, tout a été dit sur Cobol. Mais ce langage "dinosaure" reste pourtant très actif, notamment avec son patrimoine colossal de +200 milliards de ligne de codes. Il a été maintes fois attaqué, critiqué, mais il reste fidèle à lui-même. A l'automne 2016, le ministre de l'intérieur français évoque le nouveau fichier TES pour constituer un fichier national contenant les données biométiques des citoyens. Dans la présentation faite de ce projet, il a été question de moderniser le système de la carte d'identité reposant sur un code Cobol. Bref, haro sur Cobol. Bien entendu, cela a fait réagir les acteurs Cobol.

Effectivement, Cobol n'est pas obsolète, mais ce n'est pas non plus un langage d'avenir, ni moderne. Il existe toutefois de plus en plus de solutions pour injecter dans Cobol des technologies récentes et pour l'ouvrir : Cloud, open source, injection d'agilité et de DevOps, conteneurs, etc.

Le mainframe est au coeur de tout ou presque

Comme dit en ouverture de cet article, le Mainframe reste au coeur de l'informatique et de nombreuses applications business / métiers répondent à cette équation : mainframe + infrastructure distribuée + Web + mobile. Et aujourd'hui, quand on parle de modernisation du mainframe, cela passe par une approche DevOps. Ce qui parait finalement assez logique car le mainframe est un environnement de production, même si ce dernier n'est pas directement visible pour les développeurs et encore moins aux utilisateurs.

Et comme le mainframe ne va pas être jeté à la poubelle avant longtemps, mieux vaut l'intégrer aux évolutions technologiques et à l'agilité. •

DÉVELOPPEUR COBOL : PROFIL INTÉRESSANT ?

Aujourd'hui, être développeur Cobol n'est pas le profil le plus recherché. Une simple recherche sur des sites d'emplois donne des résultats très nets : -25 offres Cobol – Mainframe contre +730 pour Java ou +250 pour Python et C# ! Bref, le profil Cobol n'est pas très recherché. Mais attention, cela ne veut pas dire que la compétence Cobol ne sert à rien. Elle pourra être très utile si vous travaillez en banque / assurance ou dans certaines administrations. Elle donnera une plus-value à votre profil. Nous ne constatons pas de fortes flambées des salaires sur les annonces Cobol – Mainframe. Nous sommes dans les moyennes entre 30 et 45 000 euros selon le profil et la région. Cependant, une forte compétence Java – Mainframe va dépasser les 45 – 50 000 euros même si c'est rare. Attention aux offres très basses, nous avons vu des annonces ne dépassant pas 25 000 euros même pour une première expérience, le salaire est très bas pour le secteur.

La compétence Cobol pose un réel problème car avec les développeurs partis ou partant en retraite, les entreprises perdent de précieuses connaissances du patrimoine applicatif existant. Depuis quelques années, de nouvelles formations dédiées apparaissent notamment en université et parfois en collaboration avec les éditions mainframe.

La démat

Le mainframe est aussi vieux que l'informatique, ou presque. Tout comme pour le Cobol, on annonce sa mort depuis des années mais il est toujours là. Le patrimoine mainframe (matériel et logiciel) est considérable, et surtout stratégique pour de nombreuses entreprises. Va-t-il subir la transformation numérique et se dématérialiser ?

• François Tonic

Pour IBM, le mainframe n'est pas mort, bien au contraire. Les prochaines générations sont déjà en développement en interne et le Z14 sort courant septembre. Au-delà c'est le poids du mainframe : 36 millions de MIPS actifs(1), 6 % des serveurs Linux tournent sur un mainframe, 92 % des banques / assurances ont au moins 1 plateforme. Le poids du matériel et des systèmes pèse aujourd'hui 1,3 – 1,4 milliard $ par trimestre, dont une large proportion liée aux systèmes Z. En 2016, le mainframe avait eu une croissance de 4 % durant le dernier trimestre fiscal et les pays en pleine croissance comme la Chine sont des marchés vitaux pour les systèmes Z et les grands acteurs du mainframe.

Même si, dans l'ensemble, la partie système – matériel n'est pas la plus en forme chez IBM avec les baisses régulières des systèmes POWER et du stockage, le Z14 devrait permettre au constructeur de

(1) 1 MIPS = 1 million d'opérations par seconde

érialisation touche aussi le mainframe

booster les résultats sur les 2-3 ans à venir. Car le mainframe demeure sur des cycles longs, et les tarifs de ces machines n'ont rien à voir avec les serveurs x86 à 3 000 euros , on démarre rarement à moins de 1 million euros .
IBM n'a donc pas dit son dernier mot sur le mainframe et donc la gamme Z et le zOS. Pour le constructeur, il s'agit toujours d'un marché stratégique et cyclique (selon la sortie des nouveaux matériels). Et aujourd'hui, l'open source est très présent ainsi que de nombreuses technologiques web et x86.
Linux est poussé par IBM et avance le chiffre de 10 millions de workloads Linux déployés sur mainframe.
N'oublions pas que l'écosystème mainframe, au sens large, est actif et très large. Nous trouvons notamment les éditeurs historiques tels que Compuware, Microfocus, CA Technologies.
Pour CA, grosso modo la moitié de l'activité concerne les plateformes et systèmes mainframes. Les outils proposés sont très divers : outils de tests, collaboration, sécurité, bases de données, gestion de cycle de vie.

"Le mainframe ne se reboote jamais s'il est bien maintenu"

On redéfinit la plateforme pour tout migrer

Bien entendu tout le monde n'est pas d'accord avec la vision d'IBM et des partenaires. Par exemple TmaxSoft, éditeur coréen milite pour un système ouvert et une migration des applications et des données du mainframe vers une nouvelle plateforme entièrement logicielle.
Les arguments sont assez classiques : coût de maintenance / fonctionnement, manque de souplesse, verrouillage de la plateforme avec des éditeurs / constructeurs et difficulté à trouver les compétences nécessaires. Ces arguments sont plus ou moins la réalité, tout dépend de l'entreprise concernée. Trois solutions sont possibles : mettre à jour sa plateforme mainframe, rehosting vers une nouvelle plateforme, redéveloppement des applications.
Le rehosting (ou replatforming) est une solution technique pour déplacer tel quel un système mainframe : environnement système, applications, données. Ces solutions vont remplacer les systèmes historiques mainframe pour du Linux, et les SGBD seront modernisés (ou pas). L'objectif est de garder son environnement mais sur une plateforme moderne et non plus le socle matériel mainframe et d'utiliser des protocoles et standards actuels et ouverts. C'est ce que propose TmaxSoft avec sa solution OpenFrame. L'éditeur met en avant l'indépendance de l'infrastructure et la réduction des coûts. Toute l'opération doit se faire en transparence pour les utilisateurs. Ce type d'approche doit permettre de migrer en douceur, évite de réécrire les applications et de ré-architecturer toutes les briques techniques. Cependant, il ne faut pas en attendre de miracles surtout sur la durée des projets de migration et rien ne doit être fait dans la précipitation. Car le mainframe fait tourner et héberge des apps et des données critiques.
Après, tout dépend de la stratégie globale de l'entreprise (qui est souvent une grande entreprise ou institution / administration). Il est plus sécurisant de garder le périmètre actuel, quitte à y engouffrer une partie de son budget en maintenance et y "sacrifier" des ressources R&D sur les nouveaux services ouverts et innovants. Le poids du patrimoine (IT legacy) est toujours important. Sauf dans les startups.

Toujours le même débat : on réécrit ou on migre tel quel ?

Réécrire tout ou partie ce patrimoine d'applications et de données n'est pas non plus une mince affaire. Il sera plus facile de toucher les parties périphériques que le coeur de l'activité mainframe. Migrer des applications et SGBD non critiques peut se faire en douceur même si, là encore, il faut rester réaliste sur la durée des redéveloppements. La solution la moins radicale est de migrer les codes actuels sur des plateformes ouvertes et modernes sans réécrire. Cela nécessite des développeurs hybrides maîtrisant les technologies / outils mainframe et les technologies modernes.

L'approche OpenFrame

OpenFrame repose sur une approche 3-tiers : la partie données, la partie applications et l'interface. Le tout repose sur le système Unix / Linux. La plateforme supporte les plateformes zOS, OS390, Fujitsu, Hitachi (qui avaient leurs propres mainframes). L'objectif, comme dit précédemment, est de tout virtualiser à l'identique. Il est possible de créer des clusters OpenFrame sur le cloud (via Amazon Web Services). Vous pouvez ainsi adapter les ressources aux besoins réels nécessaires. Bien entendu, l'environnement possède un load balancer, une haute disponibilité, un système de tolérance de pannes (un mainframe ne s'arrête jamais). De nombreux outils sont disponibles : annuaire, administration, un terminal en mode web pour les écrans 3270. Pour la partie code, l'éditeur propose un compilateur compatible Eclipse.
Dans un projet de migration (type rehosting), 4 étapes sont prévues : analyse, migration, test et basculement final. La partie test doit aussi permettre de stabiliser la migration et s'assurer que tout fonctionne.
Une des tendances est le replatforming pour des raisons de coûts, de dépendances techniques. Mais dans ce genre de projets, il n'y a pas de demi-mesures. C'est tout ou rien.
L'éditeur Microfocus est en plein dans cette notion allant des besoins au déploiement.

Toujours pour pallier les manques de compétences

Comme nous l'avons déjà dit, la perte de compétences est un enjeu capital pour les entreprises, les intégrateurs et même les éditeurs. Par exemple, IBM soutient la Master Mainframe qui reçoit 10 000 participants chaque année et plus de 85 000 en tout. L'objectif est de promouvoir le mainframe dans le monde. CA Technologies promeut aussi les formations mainframe notamment avec la Mainframe Academy pour former, transmettre les connaissances et les compétences. Le risque est aussi de délocaliser le mainframe à l'externe, dans des pays qui ont les compétences.
Pour Microfocus, le profil développeur mainframe / cobol a beaucoup changé : ce sont de nouvelles générations, avec des outils modernes (Java, .Net).
Et finalement, Cobol est un langage assez facile à apprendre. Il faut regarder au-delà du compilateur. •

Aucun obstacle **pour moderniser** une application Cobol

Le marché Cobol reste dynamique et les acteurs sont là pour le prouver. Nous avons posé quelques questions à Stéphane Croce de COBOL-IT.

Il y a quelques mois, le gouvernement voyait Cobol comme une technologie obsolète. Que l'on soit d'accord ou non avec cette affirmation, cela pose tout de même la question de Cobol dans le patrimoine informatique des entreprises, et le fait que Cobol ne soit pas un langage moderne comme C++, Java, C#, etc., non ?

Le COBOL est un langage qui a été un des premiers à apparaître et qui a le plus de résilience dans le monde informatique. Il n'est pas obsolète et présente de nombreux avantages qui expliquent sa très forte présence sur le marché encore aujourd'hui. Le débat de l'obsolescence du COBOL est un faux débat, la problématique se situant plutôt au niveau du renouvellement des ressources humaines compétentes en COBOL. Le COBOL est résilient, fonctionnel et adaptable. Les langages modernes, quant à eux, sont onéreux, financièrement et humainement parlant.

Aujourd'hui, le marché mainframe n'est-il pas la maintenance, le support de l'existant et non de nouveaux déploiements ?

De notre point de vue, il faut faire attention à ne pas lier le COBOL à une plateforme – le COBOL tourne sur tout ce qui existe sur ordinateur quel que soit le constructeur, l'OS ou la marque. COBOL est portable sur tous types de serveurs et plateformes. Le mainframe n'est qu'un support pour nous. Donc si on change la question en mettant COBOL à la place de mainframe, la réponse que nous apportons est qu'aujourd'hui, on démarre rarement from scratch l'écriture d'un nouveau programme. En revanche dans un système qui existe, on développe des nouveaux projets from scratch. On ne fait pas simplement de la correction, mais on fait de la maintenance évolutive : on développe donc de nouveaux programmes en COBOL que l'on intègrera dans les architectures les plus modernes. Notons que le patrimoine COBOL continue son expansion avec plus de 5 milliards de lignes COBOL produites par an pour l'écriture de nouvelles applications. Nous pouvons également remarquer que le marché du mainframe reste encore assez actif, avec de nouvelles machines produites par IBM, même si les mainframes de petites tailles ou de taille moyenne ont tendance à disparaitre au profit de machine plus petite et fonctionnant sous Linux principalement avec un portage indéniable sur le Cloud.

Quels sont les obstacles pour moderniser, flexibiliser les codes Cobol et les environnements mainframes ?

Aucun obstacle pour moderniser le COBOL à part la formation. Si le COBOL ne se modernisait pas, il aurait déjà disparu. Le COBOL, indépendant des plateformes, a évolué avec les besoins de son temps et a répondu aux besoins fonctionnels. Le COBOL c'est une cinquantaine de verbes, simples, robustes, mais surtout fonctionnels.

Migrer telles quelles les applications Cobol sur le Cloud, des conteneurs ou une autre solution, n'est-ce pas la possibilité de retirer à terme les mainframes physiques ?

C'est une certitude que les mainframes physiques vont être remplacés par le Cloud. La seule question est : quand ? Nous sommes passés de l'ère de propriété à l'ère de l'usage. Scott McNealy disait en 1988 "Network is the power": il avait compris avant tout le monde que tout ce qui resterait de ce qu'on connaît (software, hardware, télécoms, etc.) disparaitrait au profit du réseau.

On parle maintenant de Cobol à la "mode" DevOps. N'est-ce pas deux approches totalement opposées ?

Il y a deux aspects distincts avec DevOps : 1. l'outillage, 2. l'objectif (qui est l'automatisation des tâches et l'agilité). Les outils DevOps les plus populaires aujourd'hui se marient parfaitement au COBOL : GitHub, Sonar, ou Jenkins pour ne citer qu'eux. Cette mécanique va induire plus d'agilité dans la manière de gérer ses applications avec une limite : le caractère monolithique lié à la structure même des applications COBOL d'entreprise développées à l'époque. Pour tirer pleinement parti de DevOps avec ses applications COBOL, il faudra envisager une campagne de modernisation pour « fluidifier » la maintenance et s'approcher de la nature modulaire des langages nés en même temps que DevOps.

Migrer de cobol vers un langage plus moderne : les enjeux/défis de la réécriture (où, quand, comment, avec quoi) ?

Attention à la distinction entre « migration » et « réécriture » : la migration permet un portage de l'application sur un environnement moins coûteux tout en conservant l'architecture de départ. On va, bien entendu, remplacer une base de données onéreuse par son équivalent open source, ou bien remplacer un moniteur transactionnel par un dispositif moins cher ; mais au final on conservera l'application avec son architecture d'origine. La réécriture, elle, appelle à une analyse plus profonde, rendue difficile, compte tenu de l'âge de ces applications, de la dette technique générée et souvent non maitrisée, mais surtout de la perte quasi complète des connaissances du fonctionnement de l'application. Il faut savoir que réécrire les lignes COBOL existantes demanderait 5 millions de développeurs et 65 ans. Les projets de réécriture sont souvent voués à l'échec lorsqu'ils dépassent les 200 programmes. Rien ne certifie le même niveau de service à l'arrivée, et pendant la réécriture le SI continue à vivre, imposant une gestion rigoureuse de l'ensemble (rapport de maintenance, machines en double, etc.). Tout ceci rend les projets très coûteux, avec un faible pourcentage de réussite. •

Bonnes et mauvaises pratiques de développement mobile Xamarin

Partie 2

Antony Canut
Consultant chez **Cellenza**
@AntonyCanut

Les utilisateurs de vos applications sont aussi vos clients. Ils sont très critiques sur les applications qu'ils utilisent et n'hésitent jamais à noter négativement une application qui les a déçus. Il existe bon nombre de bonnes pratiques qui sont importantes à respecter pour avoir l'application la plus réactive et la plus stable possible. Voici donc un échantillon des bonnes pratiques les plus importantes pour le développement d'une application mobile.

Utilisation du cache

La 4G est sûrement présente dans votre téléphone ; vous possédez le dernier modèle de votre constructeur favori, un smartphone surpuissant disposant de toute les dernières fonctionnalités à la mode. Cependant, vous restez un cas rare. Un smartphone peut représenter un budget très important, et le pourcentage de la population disposant d'un téléphone dernier cri est loin d'être élevé. Votre application ne pourra donc pas utiliser autant de ressources que sur votre téléphone, et le résultat pourrait être sensiblement différent. De même qu'elle est soumise au lieu où elle est lancée (pas de wifi, et peu ou pas de réseau) et quelqu'un peut choisir de l'utiliser pour récupérer une information, qu'il a peut-être déjà vue. Vous devez alors gérer le fait que l'application ne plante pas, et que l'utilisateur ait accès à tout ce qu'il est possible de lui offrir. Le cache est donc important ; il optimise également la vitesse de chargement pour des données qui ne changent pas. Dans le cas où l'utilisateur ne dispose pas d'Internet, la sauvegarde du résultat est une fonctionnalité précieuse :

```
private Dictionary<string, string> _contentCache = new Dictionary<string, string>();
private string GetAsyncUrl(string url)
{
  using (HttpClient client = new HttpClient())
  {
    try
    {
      string content = client.GetStringAsync(url).Result; // Get the response
      if (string.IsNullOrEmpty(_contentCache[url]))
          _contentCache.Add(url, content);
      return content;
    }
    catch (Exception e) // Catch exception if no internet connection / website unreachable.
    {
      return _contentCache[url] != null ? _contentCache[url] : "";
    }
  }
}
```

Dans le cas où la requête est répétée :

```
client.DefaultRequestHeaders.CacheControl.NoCache = false;
```

Ce paramètre existe par défaut sur HttpClient. Mais pour contrôler ce cache avec plus de précision, vous pouvez ajouter des paramètres à l'url pour que celle-ci change et récupère de nouvelles données.

Imbrication d'éléments graphiques

Lors de la création des vues sur Android comme sur iOS, la création d'éléments imbriqués les uns dans les autres peut avoir un effet de plus en plus négatif sur les performances. Le mieux étant sur iOS de plutôt jouer avec les contraintes pour alléger la vue de ses nombreux éléments. Sur Android, il s'agit de la même chose, tout en évitant le plus possible l'imbrication de LinearLayout qui peut largement alourdir l'interface et ses performances.

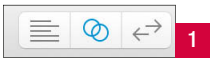
Outlets/Actions

Dans iOS, l'abonnement aux méthodes natives des boutons peut se faire de différentes manières. La plupart du temps, par méconnaissance et/ou habitude, les développeurs vont utiliser l'association par code du bouton et de sa méthode click :

```
...
 button.TouchUpInside += Button_TouchUpInside;
      }
void Button_TouchUpInside(object sender, EventArgs e)
      {

      }
```

Bien que cette manière de faire soit juste, elle n'est pas forcément optimale car, en l'état, en affichant la page plusieurs fois, l'abonnement au bouton se multiplie et crée une fuite mémoire. L'effet est encore pire avec l'utilisation d'une expression lambda.

Xamarin permet pourtant l'utilisation des Outlets (un lien vers un élément graphique) et des Actions (un lien vers un évènement d'un élément graphique) dans le développement des applications iOS. Les abonnements créés par ce biais sont automatiquement détruits lorsque l'utilisateur navigue vers une autre page. Pour cela, il faut utiliser XCode, et ouvrir le fichier xib ou storyboard, où se trouve votre élément. [1]


1

Il faut ensuite cliquer sur le bouton de l'assistant de l'éditeur parmi les trois, tout en haut à droite de l'IDE. Cela permet d'ouvrir alors une fenêtre de code suffixé ".m". Ce fichier permet d'ajouter directement des Actions sur les contrôles. Il suffira ensuite d'appuyer sur un bouton, de maintenir la touche "Ctrl" et de faire un clic puis de faire un glisser/déposer de l'élément vers le fichier ouvert [2]. A noter qu'il est également possible de diriger une action vers une autre déjà créée pour les faire toutes les deux pointer sur la même méthode. Enfin, il arrive qu'une application plante lors d'un appui sur un bouton si l'action de celui-ci n'est pas écrite dans la classe qui est sensée la référencer.

```
partial void Demo_Clicked(NSObject sender)
      {
```

```
        ...
    }
```

Il faut donc tâcher de bien écrire chaque méthode dont l'Action est référencée quitte à n'écrire son code que plus tard.

XAML Compilé

Ajouté depuis Xamarin Forms 2.0, la compilation du XAML, aussi appelée XAMLC, permet d'augmenter de manière notable les performances des applications Xamarin Forms. Elle permet d'être notifiée dès la compilation des erreurs dans le XAML plutôt qu'à son exécution. Elle permet également de réduire le poids des applications. Enfin, elle supprime les chargements et les initialisations des éléments XAML. Désactivée par défaut, la mettre en place est assez simple. Il suffit de placer un attribut dans la code-behind de la page. A noter qu'il y a également la possibilité d'appliquer ceci à toute l'application directement via un attribut global :

```
using Xamarin.Forms.Xaml;
...
[XamlCompilation (XamlCompilationOptions.Compile)]
public class MainPage : ContentPage
{
...
}
```

Abonnements (Cas des reusableCell)

Il n'est pas toujours possible de passer par des Actions sur iOS pour s'abonner à des évènements. C'est le cas par exemple des ReusableCells. Dans un exemple concret, une cellule peut contenir une méthode dans laquelle il est nécessaire de s'abonner pour réaliser une action. Or avec le scroll, la méthode permettant de créer ou de mettre à jour une cellule va être appelée plusieurs fois; l'abonnement se fera à plusieurs reprises et l'action exécutée autant de fois qu'elle contiendra d'abonnements. Ainsi la résolution se passe par ce biais :

```
cell.Selected -= Cell_Selected;
 cell.Selected += Cell_Selected;
```

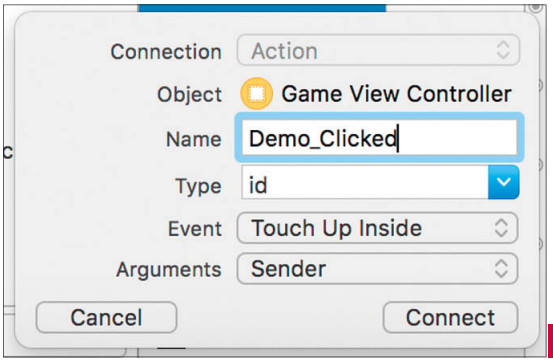
Bien que coûtant une ligne de code supplémentaire, le désabonnement permet à une cellule de ne pas appeler les méthodes auxquelles elle a pu être déjà abonnée. Sur Android et iOS, ceci est également valable lors du changement de pages pour se désabonner des méthodes pour éviter les doubles clics et les fuites mémoires.

IDisposable

Contrairement à de nombreux langages tels que le C, en .Net, le développeur n'a pas à gérer la durée de vie de ses objets. Le garbage collector passe régulièrement pour récupérer la mémoire utilisée une fois l'objet inutilisé. Cependant, certains objets ne sont pas détruits dans l'instant, et peuvent garder une trace en mémoire. Dans la mobilité, où les ressources sont réduites, l'utilisation trop élevée de la mémoire peut avoir un impact lourd sur les performances de l'application. Heureusement, certains objets du Framework .Net sont « disposables » et sont alors détruits une fois leur champ d'utilisation passé. Pour cela rien de compliqué :

```
using(var client = new HttpClient())
{
   ...
 }
```

Dans cet exemple, la variable client sera détruite et recyclée à la fin de l'accolade et la mémoire ainsi économisée. De nombreuses fuites mémoire peuvent provenir de l'oubli ou de la méconnaissance de ce type de sucre syntaxique.



2

CancellationToken

Les méthodes asynchrones peuvent, elles, donner également l'occasion de pouvoir effectuer un grand nombre de traitement en arrière-plan et ainsi de pouvoir donner des informations "en temps réel" à l'utilisateur. Mais sur un téléphone portable, de nombreux événements extérieurs peuvent perturber les traitements écrits dans vos méthodes. Ainsi, il peut arriver que le réseau se fasse plus rare ou disparaisse complètement. Si un bouton « Rafraîchir » était mis à disposition, l'utilisateur pourrait être tenté de recharger les données sans que cela ne soit forcément possible ou que la précédente méthode ne soit terminée.
Les CancellationToken, sont des moyens qui permettent à une tâche asynchrone de pouvoir être annulée en cours de traitement. Ainsi, le contrôle d'une tâche permet d'éviter de trop nombreux appels ou d'arrêter ceux qui ne peuvent aller au bout (perte de réseau). Cela permet d'économiser la batterie et, au mieux, d'informer l'utilisateur que quelque chose s'est mal passé. Cela s'utilise comme ceci :

```
bg.DoWork += (sender, args) =>
{
...
}
BackgroundWorker bg = new BackgroundWorker();
bg.WorkerSupportsCancellation = true;
CancellationTokenSource cts = new CancellationTokenSource(); CancellationToken
 token = cts.Token;
using (token.Register(() => bg.CancelAsync()))
{
bg.RunWorkerAsync();
cts.Cancel();
while (bg.IsBusy)
;
}
```

Stockage local

Une application peut avoir besoin de se connecter régulièrement à un serveur pour obtenir des données. Cependant votre utilisateur ne dispose pas toujours d'une connexion Internet. Le mieux est encore de lui afficher les précédentes données que vous pourriez avoir préalablement enregistrées. Conserver les données sur le téléphone est une marque de respect pour la batterie de votre utilisateur car, ainsi, votre application ne fait pas un nombre important de requêtes pour se mettre à jour.
Il existe avec Xamarin, de très nombreux moyens pour stocker des données dans une application, ils ne sont d'ailleurs pas tous adaptés à tous les usages. Généralement, le stockage de paramètres se fera dans les settings, celui de données volumineuses dans le système de fichiers, et celui

de données relationnelles dans une base de données. On retrouve plusieurs outils tels que SQLite, Akavache, Realm ou Couchbase Lite. Certains permettent par ailleurs la synchronisation des données avec un serveur.

Gestion des erreurs

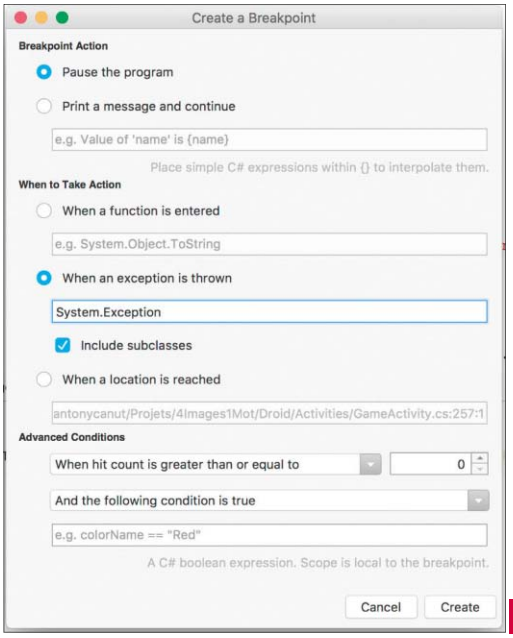
Dans une application mobile, peut-être plus que partout ailleurs, il est important de surveiller les erreurs survenant au sein de l'application. En effet, celle-ci peut planter à de nombreuses reprises durant le développement mais également ensuite lorsqu'elle est mise en production. Il est donc important de pouvoir récupérer des logs d'erreurs pour les corriger le plus rapidement possible. Une application qui plante sur le téléphone d'un utilisateur a un impact direct sur sa position dans le store.
Il existe de nombreux outils de rapport d'erreurs, d'analyse de comportements utilisateur et de déploiement tel que Mobile Center, Firebase ou Hockey App. Le mieux est toujours de faire en sorte que l'application ne plante pas et de rapporter au maximum les erreurs dans ces outils pour corriger par la suite. Il faut également noter que dans le cas de développements asynchrones, les applications peuvent planter silencieusement, sans forcément causer un arrêt immédiat de l'application mais peuvent impacter celle-ci dans ses performances. Il faut donc bien penser à relever les erreurs partout où elles apparaissent. Le mieux étant encore de développer un point d'arrêt sur toutes les exceptions de l'application : un clic droit sur la position d'un point d'arrêt pour faire apparaître une fenêtre et activer l'option qui suit [3]. Attention cependant, une application développée un peu rapidement sans prendre ceci en compte peut avoir de très nombreuses pauses dues aux multiples exceptions précédemment silencieuses. Un petit travail initial de correction est donc à prévoir mais améliore la qualité globale de l'application.

FireAndForget

Lors du développement d'une méthode sur le clic d'un bouton par exemple, les méthodes de base sont souvent synchrones. Il n'est pas possible de les rendre asynchrones. Les méthodes asynchrones que l'on appelle alors sont dites "Fire And Forget" car celles-ci sont sans retour. Le code continuera à s'exécuter sans attendre la fin de la méthode asynchrone. Une bonne pratique est d'encapsuler toute la logique dans une autre méthode qui pourra alors effectuer tout son traitement sans problème :

```
private void Button_Click(object sender, EventArgs e)
        {
            MethodAsync();
        }

private async Task MethodAsync()
        {
            await ViewModel.MethodVMAsync();
            RandomView.Alpha = 0;
            RefreshUI();
        }
```

Ainsi dans cet exemple, la méthode "RefreshUI" se trouve dans la méthode asynchrone afin de pouvoir s'exécuter avec certitude après le traitement de la méthode asynchrone contenue dans le ViewModel.
Il est également intéressant de savoir qu'une méthode "async void" est proche d'une méthode "async Task" dans le sens où elle ne dispose d'aucun retour possible mais ne peut intercepter aucune exception créant de nombreux risques d'erreurs silencieuses et de crashs sans pile d'appels.



3

Binding

Bien que le pattern MVVM soit fortement conseillé dans le développement d'applications Xamarin, il ne faut pas chercher à le faire n'importe comment pour utiliser à tout prix le Binding dans vos applications. Xamarin.Forms supporte très bien son utilisation avec ses vues et sa gestion de contexte. Mais ce n'est pas aussi bien implémenté dans la partie Native du produit.
La plupart des données existent le plus souvent en lecture seule, et ne nécessitent pas forcément de se lier avec du Binding qui pourrait alourdir légèrement les performances de l'application. Dans la plupart des cas, un abonnement sur la méthode "OnPropertyChanged" du ViewModel suffit amplement à rafraîchir les données de la vue et pourquoi pas effectuer des traitements supplémentaires. De plus, cela évite les erreurs de Binding pouvant survenir sur des éléments qui ne sont pas encore présent dans la vue.

```
private void ViewModel_PropertyChanged(object sender, System.ComponentModel.
PropertyChangedEventArgs e)
        {
            switch (e.PropertyName)
            {
                case nameof(ViewModel.PropertyWithAName):
                    ...
                    break;
            }
        }
```

CONCLUSION

Ces quelques astuces peuvent donner à votre application une sensation de fluidité plus importante. Les utilisateurs seront donc plus à même d'apprécier votre travail et de le garder dans leur smartphone. Cette valeur est d'autant plus importante que les magasins d'applications comptent dans leur algorithme de classement le nombre d'étoiles que votre application peut recevoir mais également le nombre de crashs, le nombre d'installations et de désinstallations ainsi que de nombreuses autres variables. Tâchez de toujours faire en sorte de répondre au plus de critères possibles pour atteindre le top du classement et tenter d'y rester. •

Performance et **gestion de la mémoire** avec Xamarin

Antony Canut
Développeur Xamarin
@Cellenza

cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

Avec l'expérience, on finit par comprendre qu'un téléphone est toujours moins puissant qu'un ordinateur ou un serveur. Les erreurs sont bien moins pardonnées et elles peuvent être très nombreuses. Elles impactent les performances de l'application et peuvent la ralentir fortement jusqu'à provoquer un arrêt brutal.

Les erreurs courantes

Le réseau

Pour commencer, une des erreurs les plus courantes est la mauvaise gestion des appels clients HTTP. En effet, bien souvent les développeurs ne feront pas attention aux nombreux appels réseaux et ne se méfieront pas de savoir s'il faut ou non libérer les ressources après les avoir utilisées. Il ne faut d'ailleurs pas oublier de faire les traitements en arrière-plan, pour ne pas bloquer l'application. [2]

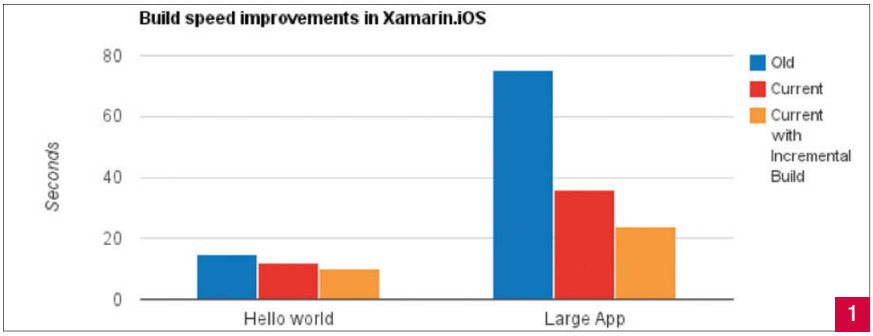
Les images

Cette erreur fait également écho à une autre avec la gestion des images, dont les ressources sont souvent utilisées mais jamais libérées pouvant créer des fuites mémoire très importantes. Il ne s'agit là que de la base, mais elle est source de beaucoup, voire de la majorité, des problèmes ; faire attention permettra à l'application de beaucoup mieux s'en sortir, même si le reste n'est pas parfait. Dans le même registre au niveau des images, il faut avoir une taille adaptée pour alléger les traitements du téléphone et au passage réduire le poids de l'application. Cette remarque est surtout valable sur Android qui risque de faire des OutOf MemoryException dans cette situation.

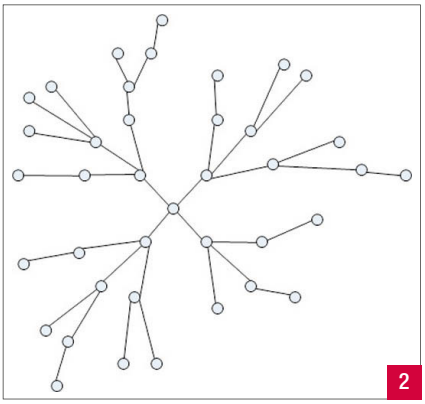
Les évènements

Une autre erreur fréquente est liée cette fois-ci aux évènements. En effet, bien souvent après avoir appris à utiliser des "expressions lambdas", un développeur aura tendance à beaucoup en utiliser pour s'abonner aux évènements. Dès lors qu'il utilise une lambda, il perd le réflexe de se désabonner de l'événement car c'est compliqué à faire.

Si l'impact d'une telle pratique sur un client lourd n'est pas très élevé, une application avec moins de ressources pourra finir par avoir de forts ralentissements ou pire. Cela est dû au fait que les expressions lambdas créent une classe qui capture les références des objets dont elle a



1



2

besoin. Si ces objets ont une durée de vie trop grande, il y a un risque de fuite de mémoire car l'expression lambda ne sera alors pas détruite non plus. L'abonnement à une méthode lors de la création d'une page a d'ailleurs le même effet si celle-ci n'est pas désabonnée lors de la navigation; elle impacte donc les performances de votre application à plus ou moins long terme.

Le matériel

Un autre point important concerne le matériel. Il ne dispose pas d'une puissance illimitée, si bien qu'au bout de quelques centaines de Mo utilisés, l'application pourra être détruite par le système pour libérer de la mémoire vive. Un développeur vigilant fera donc toujours en sorte que les traitements de son application ne prennent pas trop de ressources ou alors que ceux-ci soient découpés en sous-traitements plus courts.

De plus, chacun de ceux qui n'ont pas trait à l'interface utilisateur (Gui) doivent être exécutés en arrière-plan afin que l'application puisse toujours réagir aux interactions du système ou de l'utilisateur.

En effet, il n'y a rien de plus frustrant que d'utiliser une application qui bloque sans raison apparente. Enfin s'il n'est pas possible de faire autrement, un indicateur de chargement peut être une option sérieusement viable… pour la moindre attente.

CONCLUSION

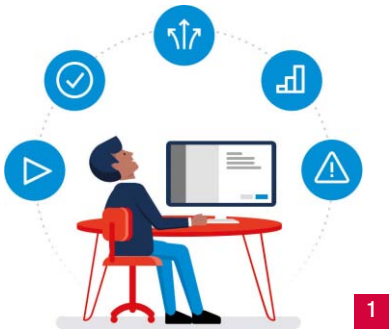
Comme souvent le plus grand problème se situe entre la chaise et le clavier alors profitez de l'été pour méditer là-dessus et, pour le bien de tous, revenez de vacances en faisant attention à vos ressources, mais aussi à celles de vos appareils. Nous avons tous besoin de vous ! •

Et le **DevOps** ?

Mathilde ROUSSEL
Consultante chez
Cellenza
Microsoft MVP
@Math_Roussel

cellenza
DOESITBETTER | Conseil - Expertise Microsoft & méthodes agiles

Aujourd'hui, le mobile a déjà assis sa domination sur de multiples domaines en informatique : nous sommes dans une ère où le développement mobile a la cote. Si, au début, les entreprises et développeurs se souciaient peu d'une stratégie à moyen terme ou long terme pour une application, c'est maintenant bien moins le cas. En effet, les grands acteurs et décideurs tels qu'Apple ou Google, bien conscients qu'un problème de qualité se posait, ont poussé les décideurs et développeurs à investir dans les démarches de qualité telles que l'intégration continue, le déploiement continu, ou plus généralement le DevOps. Pour rappel, le DevOps est une philosophie qui vise à relier les développeurs et les opérationnels pour délivrer de la qualité de façon plus efficace et plus régulière. [1]

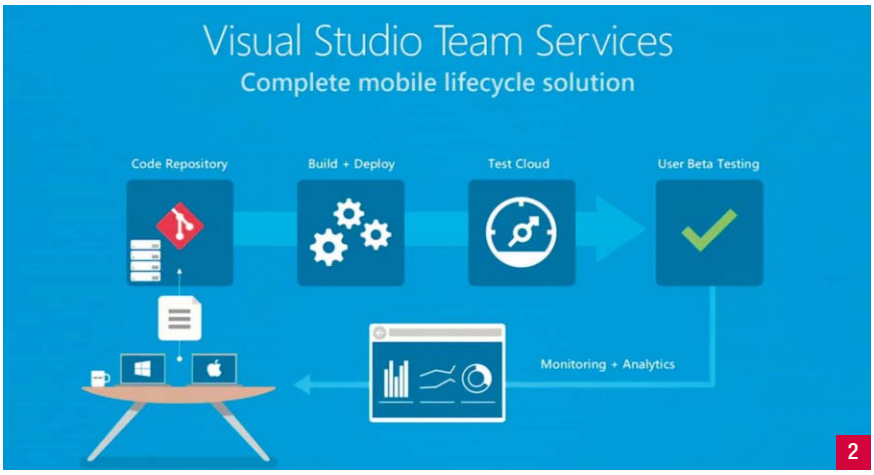


1

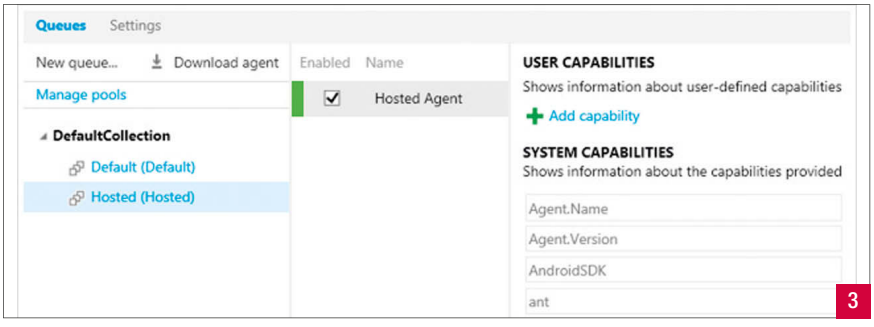
Il existe plusieurs services permettant un cycle d'intégration continue tel que TeamCity, édité par JetBrains, ou encore Jenkins. Un des acteurs majeurs actif et visible sur ce sujet est bien évidemment Microsoft qui propose depuis quelques années déjà les outils nécessaires pour répondre aux besoins du DevOps. C'est avec cette solution apportée par Microsoft que nous avons choisi de travailler pour nos projets Xamarin au sein de Cellenza.
Elle a été choisie pour plusieurs raisons. La première est que Microsoft offre une intégration complète et simple à mettre en œuvre entre les briques de DevOps. Ce premier avis se conforte par l'idée que l'ensemble des consultants Cellenza a pour cœur de métier les technologies Microsoft. La plupart ont donc déjà l'habitude d'utiliser les outils proposés par l'éditeur. Enfin, notre choix se confirme grâce à la pérennité de ces services, utilisés au quotidien par les équipes de Microsoft entre autres, et bénéficiant d'une grande communauté active.

Visual Studio Team Services, Xamarin Test Cloud, Hockey App : une symphonie ordonnée [2]

Le chef d'orchestre des outils DevOps proposés par Microsoft est Visual Studio Team Services (VSTS). Il sert non seulement de contrôleur de code source, de gestionnaire de projet agile, mais aussi de plateforme d'intégration et livraison continue. Si ce service, hébergé dans le cloud, propose une large palette d'outils, il se veut également hautement personnalisable.
Dans le cadre d'un projet Xamarin avec intégration et déploiement continu, nous allons faire un rapide focus sur les pipelines de build et de release, que nous utilisons au quotidien.



2



3

Comment construire un pipeline de build efficace ?

Lorsque l'on s'intéresse à la construction d'un pipeline de build, on se doute que tout ne va pas se faire d'un bloc. C'est surtout le cas pour une application Xamarin qui a dans la plupart des cas au moins deux plateformes cibles à builder. Partons du postulat que le but est de mettre le fruit de nos développements un jour sur les stores. Chaque store (App Store pour iOS, Google Play pour Android) a ses contraintes propres. Si le souhait est d'automatiser au maximum l'intégration continue, il est recommandé d'adopter certaines des étapes qui vont suivre. Un découpage dans les grosses mailles d'un pipeline de build pour une application Xamarin Android et iOS destinée à la publication peut-être comme suit :

- Le versionning des différentes assemblies;
- Le déchargement des données de test (qui encombreraient inutilement les packages);
- La compilation du projet Xamarin iOS;
- La compilation du projet Xamarin Android;
- Le lancement et la publication des tests unitaires;
- La signature temporaire du package Android;

- La gestion des fichiers à destination de l'artefact (ensemble de fichiers produits par le build, exploitables dès la fin du build);
- La publication des artefacts.

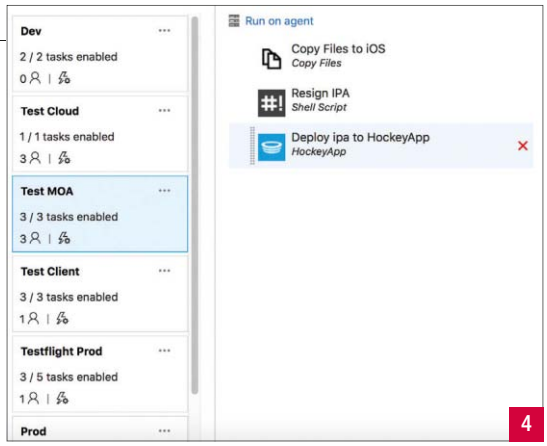
L'erreur classique réside dans l'envie de créer dès le départ un pipeline de build complet. Au contraire, il ne faut pas avoir peur de modifier de manière agile le pipeline tout au long du projet, pour répondre au mieux aux changements qui peuvent avoir lieu. En d'autres termes, il ne faut pas hésiter à adapter le pipeline de build selon le contexte du projet.

Lorsque le pipeline est configuré, se pose une nouvelle question : comment et sur quelle machine le lancer ? Visual Studio Team Services propose deux solutions à cela. La première est de laisser le service VSTS prendre en charge le build, c'est le type « Hosted ». La deuxième, appelée « default », consiste à utiliser une machine locale, configurée pour le build. [3]

L'avantage de la première solution est que la maintenance de la machine est à la charge de Microsoft. L'inconvénient qui en découle est que nous n'avons pas la main sur ces machines, et il n'est pas rare de devoir maîtriser par exemple la version du SDK Android en fonction du projet. Pour cette raison nous avons fait le choix de nous procurer des machines (des Mac), sur lesquelles des agents sont installés pour permettre de faire fonctionner les pipelines de build. Dans l'état actuel du build, l'application iOS n'est pas signée, et l'application Android ne l'est pas définitivement. Evidemment, la signature des packages n'est pas omise, mais elle se retrouve dans une autre partie de VSTS : la release.

Le build est fini, le package est prêt, que fait-on pour la release ?

La phase de release au sein de VSTS est la suite du processus DevOps et se présente sous la même forme que le build. Un pipeline de release est directement lié à un pipeline de build, puisque c'est lui qui va récupérer les artefacts publiés à la fin du build pour les traiter à son tour. Son but est, comme son nom l'indique, de fournir une nouvelle version de l'application. La différence majeure entre un pipeline de build et de release réside dans la présence d'environnements au sein de celui de ce dernier (Figure 4).



4

Ceux-ci permettent la gestion des contextes de l'application ou de l'API que consomme l'application : le développement, le test UI, le test MOA, la recette, la production…

Généralement chaque environnement possède sa propre API (éventuellement bouchonnée), ses propres jeux de données, et le pipeline de release va permettre de gérer cela. [4]

Contrairement au pipeline de build, il est plus lisible de créer un pipeline de release par plateforme cible. Cela permet une meilleure structure des environnements. Le nombre d'environnements dépend du projet, mais il est conseillé d'avoir au moins ceux cités précédemment. Dans chacun d'entre eux, plusieurs étapes clés vont entrer en jeu. La signature finale du package, ainsi que le déploiement soit sur la plateforme de recette, soit en production (donc publication sur le store). Un développement bien compartimenté et aux processus sécurisés dispose habituellement de plusieurs certificats de distribution. Le principe de signer une application mobile avec un certificat valide est primordial, car cela garantit à l'utilisateur la provenance de l'application. Cette pratique permet aussi de vérifier que l'application n'as pas été altérée entre le moment où elle sort de l'usine de release et le moment où elle est téléchargée sur le téléphone. Le fait d'avoir plusieurs certificats permet dans le cadre d'une application créée pour un client que celui-ci ait la main sur la publication sur un store. D'ailleurs sur iOS lorsqu'un certificat est créé il est obligatoire de spécifier l'usage de celui-ci, un package qui n'est pas signé avec un certificat de production ne pourra pas passer la validation Apple. Quant à Android, Google laisse plus de liberté à condition qu'une cohérence soit respectée dans l'utilisation des certificats sur le Play Store. Autrement dit, le certificat utilisé lors de la première publication sur le store devra être le même pour les mises-à-jour de l'application.


5

Afin de produire de façon automatisée les packages finaux, chaque environnement va faire appel à des scripts pour appliquer la configuration souhaitée. Par exemple, un script Powershell associé à des fichiers de configuration (au format xml ou json en fonction de l'attente de l'application) a pour but de remplacer selon l'environnement les certificats ainsi que les adresses des webservices par celles requises. Les paramètres spécifiés à ce script permettent de sélectionner le bon certificat, la bonne identité pour signer, ainsi que d'autres informations telles que l'identifiant et le nom donné à l'application. Cette manipulation évite de compiler à nouveau l'application, qui pourrait produire des binaires différents que ceux testés précédemment. Par ailleurs, l'environnement de tests UI joue lui aussi un rôle clé dans le déroulement de l'intégration continue. Afin d'assurer une meilleure qualité pour nos applications, nous réalisons des tests d'interface graphique, et pour cela nous nous appuyons sur Xamarin Test Cloud. Ce service hébergé dans le Cloud facilite les tests d'interfaces grâce à une large flotte de périphériques physiques disponibles pour réaliser les tests. Parmi les périphériques mobiles on retrouve l'iPhone, du 4 au 7 Plus, des iPads, ainsi que des Samsung, Nexus, etc. Chaque téléphone ou tablette existe avec des versions du système différentes afin de tester l'application dans le plus de conditions possibles. Enfin, pour le déploiement vers les plateformes de bêta, nous avons choisi d'utiliser HockeyApp qui est également un produit Microsoft, et qui gère les trois plateformes mobiles principales. Pour chaque environnement une application a été créée, avec pour chacune le groupe de distribution adéquat. [5]

CONCLUSION

Cet article avait pour vocation de présenter un ensemble de bonnes pratiques DevOps que nous mettons en place pour les projets Xamarin de nos clients. Forts de nos apprentissages nous avons réussi à architecturer des processus efficaces pour l'intégration et le déploiement continu des applications mobiles. Ces processus et outils mis en œuvre garantissent un niveau de qualité toujours meilleur. De plus nous pouvons compter sur les nombreuses nouveautés apportées régulièrement par les équipes Microsoft qui nous permettent de nous perfectionner en permanence. L'avenir se tourne d'ailleurs vers un nouvel outil : Visual Studio Mobile Center. Celui-ci regroupe l'ensemble des services proposés par VSTS, Xamarin Test Cloud ou HockeyApp sur une seule plateforme. •

Tous les numéros de

PROGRAMMEZ!

le magazine des développeurs

sur une clé USB (depuis le n°100)

34,99 €*

Clé USB. Photo non contractuelle. Testé sur Linux, OS X, Windows. Les magazines sont au format PDF.

** tarif pour l'Europe uniquement. Pour les autres pays, voir la boutique en ligne*

Commandez la directement sur notre site internet : www.programmez.com

Complétez votre collection

Prix unitaire : ***6,50€***

☐ 176	: |__| exemplaire(s)	☐ 200	: |__| exemplaire(s)
☐ 177	: |__| exemplaire(s)	☐ 204	: |__| exemplaire(s)
☐ 181	: |__| exemplaire(s)	☐ 206	: |__| exemplaire(s)
☐ 191	: |__| exemplaire(s)	☐ 209	: |__| exemplaire(s)
☐ 193	: |__| exemplaire(s)		

Commande à envoyer à :
Programmez!
57 rue de Gisors - 95300 Pontoise

Prix unitaire : 6,50 € (Frais postaux inclus)

soit \|__\|__\| exemplaires x 6,50 € = \|__\|__\|__\|__\| € soit au **TOTAL** = \|__\|__\|__\|__\|__\| €

☐ M. ☐ Mme Entreprise : \|__\|__\|__\|__\|__\|__\|__\|__\| Fonction : \|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|

Prénom : \|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\| Nom : \|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|

Adresse : \|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|

Code postal : \|__\|__\|__\|__\|__\| Ville : \|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|__\|

Règlement par chèque à l'ordre de Programmez !

Offres pouvant s'arrêter à tout moment, sans préavis

PROG 210

Intelligence Artificielle : découverte des Algorithmes Génétiques en Java Partie 1

• Sylvain Saurel
sylvain.saurel@gmail.com
Développeur Android
https://www.ssaurel.com

Depuis de nombreux mois désormais, l'Intelligence Artificielle (IA) est devenue un sujet grand public. Il ne se passe pas une semaine sans que les médias n'en parlent comme la bataille du futur (et déjà du présent) entre les géants Américains que sont Google, Apple ou encore Microsoft. L'Intelligence Artificielle constituera bien un enjeu majeur dans le futur de l'informatique et dans cet article, nous vous proposons de découvrir les Algorithmes Génétiques qui sont un des nombreux outils que met à disposition l'IA aux développeurs.

L'Intelligence Artificielle est devenue un sujet de buzz depuis plusieurs mois au point que le grand public est désormais conscient de l'existence de cette discipline et commence à s'inquiéter de la puissance qu'elle pourra conférer à des ordinateurs et des robots dans le futur. Du point de vue du développeur, l'Intelligence Artificielle est une discipline passionnante aux domaines d'application divers et variés. Dans cet article, nous allons nous intéresser plus particulièrement aux Algorithmes Génétiques qui constituent un des nombreux outils du domaine de l'IA.


1
Les AG sont un outil puissant de l'IA

Que sont les Algorithmes Génétiques ?

Avant tout, il convient de définir précisément ce que sont les Algorithmes Génétiques. Il s'agit d'une technique de recherche et d'optimisation métaheuristique basée sur des principes présents dans la théorie de la sélection naturelle. Ils visent à obtenir une solution approchée à un problème d'optimisation lorsque la solution est inconnue ou qu'aucune méthode exacte de résolution en un temps raisonnable n'est connue. [1]

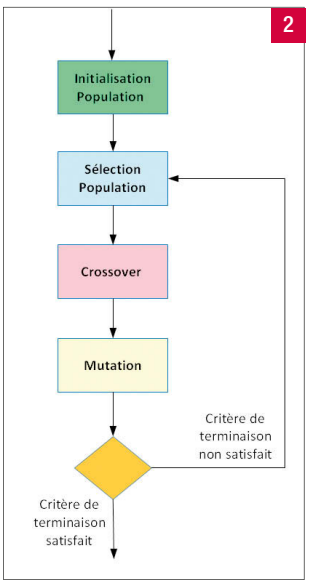
Un peu de Théorie

Les Algorithmes Génétiques étant basés sur des phénomènes biologiques, ils reprennent une grande partie des termes du domaine de la génétique. De fait, on va travailler sur des populations d'individus ayant des chromosomes qui sont des chaînes d'ADN. Les Algorithmes Génétiques vont ensuite simuler 3 grands processus évolutionnaires : la sélection, le crossover ou croisement des chaînes d'ADN entre chromosomes et enfin la mutation qui peut survenir de manière aléatoire au sein d'un chromosome. La sélection quant à elle se base sur la théorie de l'évolution qui avance, qu'au fil du temps, les gènes conservés au sein d'une population sont ceux qui sont les plus adaptés aux besoins de l'espèce en regard de son environnement.

Le cœur d'un Algorithme Génétique est sa fonction mesurant l'adaptation d'un individu à son environnement. En transposant à l'informatique, l'environnement correspondra aux contraintes s'appliquant à la solution à trouver ou à approximer pour un problème, et l'individu correspondra à la solution la plus adaptée à ce problème. De fait, un individu aura autant de gènes dans son chromosome que de paramètres d'entrée pour un problème donné. La manière d'encoder les gènes d'un chromosome pour un individu sera détaillée plus tard dans cet article. Au niveau du fonctionnement d'un Algorithme Génétique, on part d'une population d'individus ayant au moins un chromosome et une valeur définissant le niveau d'adaptabilité d'un individu par rapport à son environnement. Plus concrètement, cette valeur définira la qualité de la solution par rapport au problème considéré. Au démarrage de l'algorithme, on va générer aléatoirement une population d'individus d'une taille donnée. On évalue ensuite l'adaptabilité des individus de la population en regard du problème dans ce qui précède la fameuse étape de sélection. Si une solution acceptable (ou exacte suivant le problème) est trouvée, on s'arrête là. Dans le cas contraire, on rentre dans une boucle permettant de produire de nouvelles générations d'individus. Au sein de la boucle de génération d'individus, on applique tout d'abord l'opération de crossover sur la population. Le pourcentage de croisements à réaliser à chaque génération étant un paramètre d'entrée de l'Algorithme Génétique. La seconde étape consiste à appliquer un certain nombre de mutations sur les chromosomes des individus de la population en se basant sur le pourcentage de mutation défini en entrée. Enfin, on évalue la population pour ne garder que les éléments de la génération courante les plus adaptés et pour vérifier si une solution acceptable a été trouvée sans oublier d'incrémenter l'index de génération. En effet, puisque dans de nombreux cas on ne pourra trouver qu'une solution la plus optimisée possible, on définira en entrée de l'Algorithme Génétique le nombre de générations au bout duquel l'algorithme se terminera. Ce fonctionnement global des Algorithmes Génétiques est résumé au sein de la figure [2].

Place à la pratique !

Les fondements théoriques des Algorithmes Génétiques présentés, il est temps de passer à la pratique en implémentant un Algorithme Génétique en Java pour résoudre le problème suivant : considérant une population d'individus ayant un chromosome constitué de gènes pouvant prendre comme valeur 0 ou 1, le but va être d'obtenir un individu dont tous les gènes d'un chromosome ont pour valeur 1. Volontairement simpliste, ce problème nous donnera l'occasion d'implémenter le cadre de résolution commun à tous les Algorithmes Génétiques qui a été présenté dans la partie théorique.



Corps d'un Algorithme Génétique

Création d'un Individu

En premier lieu, nous allons modéliser un individu de notre population au sein d'une classe Individual contenant un tableau d'entiers représentant les gènes du chromosome ainsi qu'un champ fitness définissant l'optimalité de l'individu en regard de la solution visée. Dans le cadre de notre problème, la valeur de la propriété fitness sera maximum lorsque tous les gènes du chromosome de l'individu seront valorisés à 1. Le constructeur de la classe Individual permet de définir le nombre de gènes du chromosome à créer ainsi que la valeur des gènes définie aléatoirement à 0 ou à 1 ce qui nous donne le code suivant :

```
public class Individual {
  private int[] chromosome;
  private double fitness = -1;

  public Individual(int[] chro) {
    chromosome = chro;
  }

  public Individual(int chromosomeLength) {
    chromosome = new int[chromosomeLength];

    for (int gene = 0; gene < chromosomeLength; gene++) {
      setGene(gene, 0.5 < Math.random() ? 1 : 0);
    }
  }
  // Getters et Setters ...
}
```

Création de la Population

Nous passons ensuite à la modélisation de la population au sein d'une classe éponyme Population composée d'un ensemble d'objets Individual et d'une propriété populationFitness. La classe Population sera utilisée pour réaliser des opérations communes à l'ensemble des individus comme trouver l'individu le plus adapté ou encore sélectionner les individus pour les opérations de mutation et de crossover. Elle aura l'allure suivante :

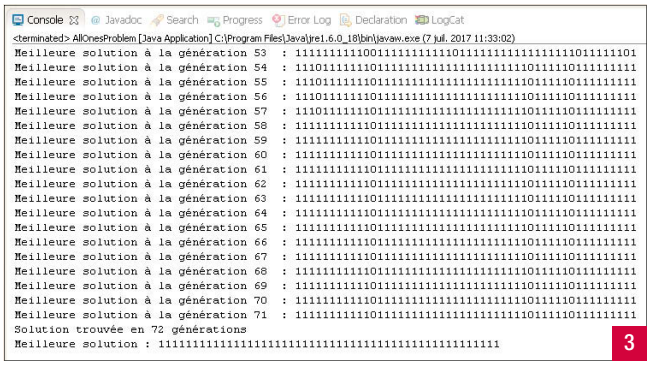
```
public class Population {
  private Individual population[];
  private double populationFitness = -1;

  public Population(int populationSize) {
    population = new Individual[populationSize];
  }

  public Population(int populationSize, int chromosomeLength) {
    population = new Individual[populationSize];

    // Création des individus
    for (int individualCount = 0; individualCount < populationSize; individualCount++) {
      population[individualCount] = new Individual(chromosomeLength);
    }
  }

  // Getters et Setters ...

  // Retourne l'individu le plus adapté situé à la position offset
  public Individual getFittest(int offset) {
    // Tri de la population par valeur d'adaptabilité
    Arrays.sort(population, new Comparator<Individual>() {
      @Override
      public int compare(Individual o1, Individual o2) {
  if (o1.getFitness() > o2.getFitness()) {
    return -1;
  } else if (o1.getFitness() < o2.getFitness()) {
    return 1;
}

          return 0;
        }
      });

    return population[offset];
  }

  // Brassage de la population
  public void shuffle() {
    Random r = new Random();

    for (int i = population.length - 1; i > 0; i--) {
      int index = r.nextInt(i + 1);
      Individual a = population[index];
      population[index] = population[i];
      population[i] = a;
    }
  }
}
```

Abstraction de notre Algorithme Génétique

Afin de regrouper les opérations effectuées au sein de notre Algorithme Génétique telles que la mutation, le crossover ou la sélection des individus, nous allons créer une classe GeneticAlgorithm possédant plusieurs propriétés telles que le pourcentage de mutation et de crossover ou encore le nombre d'éléments faisant partie de l'élite de la population et qui seront de fait préservés lors des opérations de mutation et de crossover.
La classe GeneticAlgorithm possède également une méthode calcFitness prenant en entrée un objet Individual et retournant son niveau d'adaptabilité. Appliqué à notre problème, cela revient à compter le nombre de gènes à 1 au sein du chromosome de l'individu puis à diviser cette valeur par le nombre de gènes total pour obtenir une valeur flottante comprise entre 0 et 1. Une méthode evalPopulation nous assurera que chaque individu d'une population est bien évalué mais également d'obtenir un score global d'adaptabilité pour la population.
Pour éviter que notre algorithme boucle à l'infini, il faut définir une condition de terminaison. Ici, cela consiste à vérifier qu'au moins un individu a un niveau d'adaptabilité égal à 1 au sein d'une méthode isTerminationConditionMet. En sus, une méthode selectParent retournera un élément candidat au crossover.
Il reste maintenant à définir une méthode crossoverPopulation chargée d'effectuer l'opération de crossover sur une génération donnée d'individus de la population en tenant compte du pourcentage de croisements à réaliser mais également en préservant les éléments les plus prometteurs. Pour le croisement, on va simplement créer un nouvel individu ayant pour gènes la moitié des gènes du premier parent et la moitié des gènes

du second. Cette méthode ne nous garantit absolument pas d'obtenir un individu plus adapté suite au crossover mais cela s'avère suffisant dans le cadre de notre problème. Un crossover plus performant pouvant être mis en place pour des problèmes plus complexes.
Enfin, nous terminons en définissant la méthode mutatePopulation servant à appliquer des mutations sur les individus les moins prometteurs de la population. Pour ces individus, on parcourt ainsi l'ensemble des gènes dans le but d'appliquer un certain nombre de mutations, c'est-à-dire inverser la valeur d'un gène ici, en tenant compte du pourcentage de mutations défini en entrée. Il reste ensuite à ajouter l'individu mutant à la population de la génération suivante. Tout ceci nous donne le code suivant pour la classe GeneticAlgorithm :

```
public class GeneticAlgorithm {
  private int populationSize;
  private double mutationRate;
  private double crossoverRate;
  private int elitismCount;

  public GeneticAlgorithm(int populationSize, double mutationRate, double
 crossoverRate, int elitismCount) {
    this.populationSize = populationSize;
    this.mutationRate = mutationRate;
    this.crossoverRate = crossoverRate;
    this.elitismCount = elitismCount;
  }

  public Population initPopulation(int chromosomeLength) {
    Population population = new Population(populationSize, chromosomeLength);
    return population;
  }

  public double calcFitness(Individual individual) {
    int correctGenes = 0;

    for (int geneIndex = 0; geneIndex < individual.getChromosomeLength(); geneIndex++) {
      // Ajout de 1 si un gène est correct
      if (individual.getGene(geneIndex) == 1) {
        correctGenes += 1;
      }
    }

    // Calcul de la valeur d'adaptabilité
    double fitness = (double) correctGenes / individual.getChromosomeLength();
    individual.setFitness(fitness);
    return fitness;
  }

  public void evalPopulation(Population population) {
    double populationFitness = 0;

    // Parcours de la population et calcul du niveau d'adaptabilité
    for (Individual individual : population.getIndividuals()) {
      populationFitness += calcFitness(individual);
    }

    population.setPopulationFitness(populationFitness);
  }

  public boolean isTerminationConditionMet(Population population) {
    for (Individual individual : population.getIndividuals()) {
      if (individual.getFitness() == 1) {
        return true;
      }
    }

    return false;
  }

  public Individual selectParent(Population population) {
    Individual individuals[] = population.getIndividuals();
    double populationFitness = population.getPopulationFitness();
    double rouletteWheelPosition = Math.random() * populationFitness;

    // Recherche d'un parent
    double spinWheel = 0;

    for (Individual individual : individuals) {
      spinWheel += individual.getFitness();

      if (spinWheel >= rouletteWheelPosition) {
        return individual;
      }
    }

    return individuals[population.size() - 1];
  }

  public Population crossoverPopulation(Population population) {
    // Création de la nouvelle population
    Population newPopulation = new Population(population.size());

    for (int populationIndex = 0; populationIndex < population.size(); populationIndex++) {
      Individual parent1 = population.getFittest(populationIndex);

      // Critère d'application du crossover
      if (crossoverRate > Math.random() && populationIndex >= elitismCount) {
        Individual offspring = new Individual(parent1.getChromosomeLength());
        Individual parent2 = selectParent(population);

        for (int geneIndex = 0; geneIndex < parent1.getChromosomeLength(); geneIndex++) {
          // Utilisation moitié gènes parent 1 et parent 2
          offspring.setGene(geneIndex, 0.5 > Math.random() ?
            parent1.getGene(geneIndex) : parent2.getGene(geneIndex));
        }

        newPopulation.setIndividual(populationIndex, offspring);
      } else {
        newPopulation.setIndividual(populationIndex, parent1);
      }
    }

    return newPopulation;
  }
```



Solution trouvée avec notre AG

```
public Population mutatePopulation(Population population) {
 Population newPopulation = new Population(populationSize);

 for (int populationIndex = 0; populationIndex < population.size(); populationIndex++) {
  Individual individual = population.getFittest(populationIndex);

  for (int geneIndex = 0; geneIndex < individual.getChromosomeLength(); geneIndex++) {
   // On préserve les meilleurs individus des mutations
   if (populationIndex > elitismCount) {
    if (mutationRate > Math.random()) {
     int newGene = individual.getGene(geneIndex) == 1 ? 0 : 1;
     individual.setGene(geneIndex, newGene);
    }
   }
  }

  newPopulation.setIndividual(populationIndex, individual);
 }

 return newPopulation;
 }
}
```

Corps de l'Algorithme Génétique

Les différentes parties de notre Algorithme Génétique créées, il est temps de les faire fonctionner de concert pour résoudre notre problème. Nous créons une instance de la classe GeneticAlgorithm en passant en entrée les paramètres suivants :

- Le nombre d'individus de la population ;
- Le pourcentage de mutations à appliquer lors de la création de chaque génération d'individus ;
- Le pourcentage de croisements à appliquer ;
- La taille de l'élite, c'est-à-dire le nombre d'individus les plus prometteurs qui ne seront pas affectés par le crossover ou les mutations ;

On initialise ensuite la population d'individus via appel à la méthode initPopulation de la classe GeneticAlgorithm en passant en entrée le nombre de gènes du chromosome de chaque individu. On effectue une première évaluation du niveau d'adaptabilité de la population en appelant la méthode evalPopulation. On rentre alors dans la boucle de création de générations présentée à la figure [2]. La condition de sortie étant déterminée par la méthode isTerminationConditionMet de la classe GeneticAlgorithm. Au sein de cette boucle, on applique d'abord l'opération de crossover sur la population via appel à crossoverPopulation puis l'opération de mutation via appel à mutatePopulation. Il est alors temps d'évaluer l'adaptabilité de la nouvelle population d'individus générée avant que la condition de terminaison soit à nouveau testée ce qui nous amène au code suivant :

```
public class AllOnesProblem {

 public static void main(String[] args) {
  // Création de l'instance de GeneticAlgorithm
  GeneticAlgorithm ga = new GeneticAlgorithm(100, 0.001, 0.95, 2);
  // Initialisation population
  Population population = ga.initPopulation(50);
  // Evaluation
  ga.evalPopulation(population);

  // Index génération
  int generation = 1;

  // Boucle de création
  while (!ga.isTerminationConditionMet(population)) {
   System.out.println("Meilleure solution à la génération " + generation + " : " +
population.getFittest(0).toString());

   // Crossover
   population = ga.crossoverPopulation(population);

   // Mutation
   population = ga.mutatePopulation(population);

   // Evaluation
   ga.evalPopulation(population);

   // Nouvelle génération
   generation++;
  }

  System.out.println("Solution trouvée en " + generation + " générations");
  System.out.println("Meilleure solution : " + population.getFittest(0).toString());
 }
}
```

Exécution de l'Algorithme Génétique

Notre Algorithme Génétique en place pour la résolution de notre problème, il est temps de passer à son exécution afin de vérifier son bon fonctionnement. Le résultat de l'exécution de notre Algorithme Génétique, présenté à la figure [3], montre que nous obtenons une solution à notre problème au bout de 72 générations avec un individu dont tous les gènes ont pour valeur 1. Bien entendu, le nombre de générations avant d'obtenir une solution au problème variera entre chaque exécution puisque la création de l'individu optimal est dépendante des mutations et des croisements qui sont appliqués de manière aléatoire.

CONCLUSION

Les Algorithmes Génétiques constituent un outil puissant du domaine de l'Intelligence Artificielle qu'il est important de connaître et de maîtriser. Leur application à des problèmes de recherche et d'optimisation donne souvent de bons résultats lorsque la solution est inconnue ou qu'aucune méthode exacte de résolution en un temps raisonnable ne fonctionne. L'application à la résolution d'un premier problème trivial aura permis de prendre en main leur implémentation en Java. Dans la seconde partie de cet article, nous verrons comment résoudre le célèbre problème du Voyageur de Commerce à l'aide des Algorithmes Génétiques. Vaste programme en perspective ! •

Créer une API Hypermedia avec Threerest et Node.js en moins de 10 minutes

Partie 2

Vincent Piard
Expert Technique
SQLI Nantes

Wilfried Moulin
Expert Technique
SQLI Nantes.

Une API (Application Programming Interface) ou interface de programmation est un ensemble de normes qui permet l'échange de données entre des systèmes hétérogènes. En clair, il s'agit d'un code qui remplit un service. Il peut donc être sollicité par n'importe quel acteur (humain ou machine). Donc pour une API donnée, on aura un fournisseur de service autrement dit l'API et le consommateur du service, un client Web par exemple. La beauté des API réside dans le fait que le consommateur ne connaît pas l'implémentation du service. De ce fait, un client PHP peut consommer une API en PHP, en Java ou bien encore en Go. La seule contrainte pour l'API est de rendre correctement son service.

Le combat de l'implémentation

Pour notre BDthèque cela va se manifester par la possibilité à partir d'un auteur de naviguer vers ses séries puis vers les titres des séries. La navigation se faisant dans un sens bidirectionnel.

L'hypermédia a de nombreuses spécifications (Siren, Hal, Json-API, Hydra, Json-LD…). Actuellement, cette abondance de spécifications est un frein au développement de l'hypermédia. Ces spécifications servent à fournir un format commun d'échange et de représentations des liens permettant ainsi l'interopérabilité plus aisée. Il est donc primordial de bien s'accorder sur la spécification à utiliser.

Le format HAL est le plus globalement adopté et se démarque de ses concurrents par sa simplicité. De plus, la communauté HAL est de loin la plus importante, il sera donc plus facile de trouver ressource et aide auprès de celle-ci. Nous allons donc nous fier à cette spécification afin d'implémenter notre API hypermédia. Rassurez-vous, globalement la différence porte sur la façon d'implémenter nos liens, donc la marche entre les différentes spécifications n'est pas énorme.

Une ressource auteur de BD ressemble à ceci :

```
{
    "firstName": "Denis",
    "id": "3",
    "lastName": "Bajram",
    "series": [
      {
        "idSerie": 6,
        "name": "Expérience mort"
      },
      {
        "idSerie": 7,
        "name": "Universal War One"
      },
      {
        "idSerie": 8,
        "name": "Universal War Two"
      }
    ]
  }
}
```

L'équivalent en mode "HAL" correspond au flux Json suivant :

```
{
  "_links": {
    "self": {
      "href": "/authors/3"
    }
  },
  "data": {
    "_links": {
      "self": {
        "href": "/authors/3"
      }
    },
    "firstName": "Denis",
    "id": "3",
    "lastName": "Bajram",
    "series": [
      {
        "_links": {
          "self": {
            "href": "/series/6"
          }
        },
        "idSerie": 6,
        "name": "Expérience mort"
      },
      {
        "_links": {
          "self": {
            "href": "/series/7"
          }
        },
        "idSerie": 7,
        "name": "Universal War One"
```

```
        },
        {
          "_links": {
            "self": {
              "href": "/series/8"
            }
          },
          "idSerie": 8,
          "name": "Universal War Two"
        }
      ]
    }
}
```

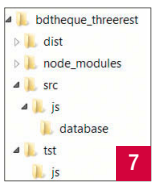
Pour vous présenter et simplifier tout ça, nous allons nous appuyer sur le Framework threerest. Il est issu d'une idée pour un autre projet, et créé en étroite collaboration avec Wilfried Moulin, Expert technique à SQLI Nantes. En gros, il permet de créer des services REST de niveau 2 et peut gérer des notions d'hypermédia.

Donc le Framework threerest existe et va nous aider dans cette tâche. Les sources sont disponibles sous https://github.com/wmoulin/threerest. Ce Framework est basé sur les décorateurs (services REST, méthode HTTP, hypermédia...). Si vous ignorez ce que sont les décorateurs en JS, je vous renvoie vers l'article suivant : https://blog.hadrien.eu/2015/06/05/decorateurs-es7/.

Il faudra donc travailler sur une version de spécification d'EcmaScript pas encore implémentée dans NodeJs et par aucun navigateur (au moment de la rédaction de l'article). De plus, dans nos différents exemples, nous allons utiliser la syntaxe de module et de classe ES6. Pour ce faire, nous allons donc devoir transpiler le code en une version d'EcmaScript supportée par ce dernier. Là aussi pour nous simplifier la tâche, nous nous sommes appuyés sur un outil basé sur Gulp et créé par Wilfried Moulin : Joobster. Joobster est un facilitateur qui a donc pour but de faciliter la vie du développeur JS en prenant en charge les parties de transpilation, de test et de packaging. Ce facilitateur est en cours de construction et pour l'instant quelques premières fonctionnalités sont implémentées, mais cela va suffire à notre bonheur. Pour l'installer, il suffit de taper l'instruction suivante :

```
npm install joobster -g
```

Cela va installer globalement cet utilitaire. Pour pouvoir gérer plus facilement la transpilation, ce framework se base sur une structure prédéfinie. Il est possible d'utiliser une structure propre, mais pour cela il faudra la préciser dans un fichier joobster.json à la racine du projet.



7

Nous allons donc créer un nouveau projet nommé bdtheque_threerest. La structure va être la suivante : [7]

Vos sources seront dans src et tst et votre code transpilé sera recopié dans dist. Le code à exécuter sera donc dans dist. Voici le contenu du fichier joobster.json :

```
{
  "javascript": {
    "compile" : {
      "presets": ["es2015", "stage-1"],
      "plugins" : ["transform-decorators-legacy"]
    }
  }
}
```

Sans trop rentrer dans les détails, voici la liste des commandes que nous allons utiliser :

- **jsr clean:js** pour supprimer le résultat de la transpilation contenu dans dist/js ;
- **jsr compile:js** pour transpiler le code contenu dans src/js vers dist/js (avec en dépendance la tache de nettoyage) en utilisant Babel avec les options précisées dans le fichier de configuration.

Pour installer threerest nous allons l'ajouter au package.json :

```
{
  "name": "bdtheque_threerest",
  "description": "mon application de demo pour une gestion des BDs avec le framework threerest",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "express": "^4.13.4",
    "threerest": "1.0.4"
  },
  "scripts": {
    "start": "jsr compile:js & node ./dist/js/app"
  }
}
```

Il est également possible de le faire de manière automatique avec la commande

```
npm install threerest --save
```

Il suffit alors, de faire un npm install et voici express et threerest installés dans votre application.

Veuillez noter la balise scripts dans le package.json. Cette balise permet de lancer des scripts. Ici le script start va compiler votre projet puis va lancer l'app transpilée. Il faudra donc taper npm start et non plus node app.js pour lancer votre serveur node.

Afin de fonctionner pour le mieux nous allons découper nos services par unité fonctionnelle. Nous allons donc ajouter un répertoire services au même niveau que database. Nous allons aussi ajouter un fichier vide nommé serviceTest.js. Pour que celui-ci fonctionne, nous allons importer les spécificités de threerest.

```
import { Service } from "threerest";
import { Methods } from "threerest";
import { Hal } from "threerest";
```

Le décorateur Service permet d'indiquer que telle classe représente tel service, par exemple :

```
@Service.path("/test")
export default class ServiceTest {
...
}
```

Ici la classe ServiceTest sera le service pour la ressource test. Cette classe va nous permettre de voir le fonctionnement de base de threerest.

Les décorateurs Methods permettent d'associer une méthode de notre service à un verbe HTTP sur l'url de notre ressource. Il est donc également possible de déclarer un complément de route au sein de ce décorateur. Par exemple :

```
@Methods.get("/")
```

```
test() {
  return "it's work !!!";
}
```

Donc l'URI pour obtenir la ressource test sera /test.
Le dernier décorateur, mais pas le moins important, est Hal. Il va permettre de rendre son API hypermedia en ajoutant des liens sur son service.

```
@Methods.get("/")
@Hal.halServiceMethod()
test() {
  return "it's works !!!";
}
```

Dernière manipulation mais non des moindres à faire avant de lancer notre service. Il faut référencer celui-ci auprès de l'application afin de l'instancier. La manipulation se fait dans le fichier app.js.

```
import * as ServiceTest from "./services/serviceTest";
...
threerest.ServiceLoader.loadService(app, new ServiceTest.default());
```

Pour rappel, la commande pour lancer votre serveur sera cette fois, npm start. La réponse pour un service qui possède ces trois décorateurs (url http:\\localhost:8080\test) sera la suivante :

```
{
   "_links": {
      "self": {
         "href": "/test"
      }
   },
   "data": "it's work !!!"
}
```

Si l'on utilise les décorateurs Hal de threerest, la réponse du service sera décorée de la manière suivante. Le résultat brut du service est mis dans la partie data et la partie link porte les liens hypermedia. Dans le cas contraire, le service revoit la valeur directement.
Donc nous avons maintenant créé un service qui nous renvoie notre donnée ainsi qu'un unique lien nous indiquant où l'on se trouve. Allons un peu plus loin à la découverte de ce framework. Chaque ressource de vos API va faire l'objet d'une classe JS. Prenez l'exemple suivant :

```
@Hal.halEntity("/authors/:id")
class Author {

  @Hal.resourceId()
  id = 0;

  constructor(id, firstName, lastName, series) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.series = series;
  }
}
```

Le décorateur @Hal.halEntity permet de définir la classe Author comme une entité du modèle Hal. Le chemin entre parenthèses permet de définir la route vers la ressource.
Le décorateur @Hal.resourceId indique l'identifiant à utiliser pour gérer le lien de cette ressource. Il est donc important de ne pas oublier ce décorateur.
L'intérêt de déclarer les objets métier de cette manière va être de pouvoir générer les liens pour l'ensemble des ressources/sous-ressources de votre API. Donc il va falloir créer un objet Author, un Serie et un Title pour chaque ressource de notre Bdthèque.
Il est maintenant temps de créer notre service, qui va nous renvoyer une liste d'auteurs. Ce sera fait dans le fichier serviceAuthors. N'oubliez pas d'instancier votre service dans l'app.js comme précédemment.

```
@Methods.get("/:id")
@Hal.halServiceMethod()
getSwitchId(value) {
  var id = value.id;
  var result = BdHelper.searchParams(db, 'authors', 'id', id);
  if (result) {
     return BdHelper.getAuthor(result, id);
  }
  throw new NotFoundError();
}
```

L'interrogation de la ressource /authors/2 va nous renvoyer le flux suivant :

```
{
   "_links": {
      "self": {
         "href": "/authors/2"
      }
   },
   "data": {
      "_links": {
         "self": {
            "href": "/authors/2"
         }
      },
      "firstName": "Christophe",
      "id": "2",
      "lastName": "Bec",
      "series": [
         {
            "_links": {
               "self": {
                  "href": "/series/1"
               }
            },
            "idSerie": 1,
            "name": "Carthago"
         },
         {
            "_links": {
               "self": {
                  "href": "/series/2"
               }
            },
```

```
            "idSerie": 2,
            "name": "Deepwater Prison"
        },
        {
            "_links": {
                "self": {
                    "href": "/series/3"
                }
            },
            "idSerie": 3,
            "name": "Meilleur Job du monde (le)"
        },
        {
            "_links": {
                "self": {
                    "href": "/series/4"
                }
            },
            "idSerie": 4,
            "name": "Prométhée"
        },
        {
            "_links": {
                "self": {
                    "href": "/series/5"
                }
            },
            "idSerie": 5,
            "name": "Temps des loups (Le)"
        }
    ]
  }
}
```

Afin d'obtenir le résultat suivant n'oubliez pas d'annoter votre classe Serie comme votre classe Author.

La pagination lui tombe sur la tête

Threerest permet aussi de gérer la pagination de façon simple. Il suffit d'ajouter un boolean à true au sein du décorateur @Hal.halServiceMethod(). Donc si votre service renvoie une liste celle-ci sera automatiquement paginée. Par défaut, threerest se base sur les mots clés pageSize et pageIdx. Le mot clé pageSize indique le nombre de résultat que le service doit renvoyer. PageIdx indique la position de début dans la liste. Naturellement si vous souhaitez utiliser d'autres mots clés, il est tout à fait possible de les customiser en les précisant dans le décorateur. L'exemple porte sur la liste des titres de la base de données. Voici la déclaration par défaut :

```
@Service.path("/titles")
export default class ServiceTitles {

 @Methods.get("/")
 @Hal.halServiceMethod(true)
 getAll() {
  return BdHelper.getTitles(db);
 }
}
```

Et celle en spécifiant les mots clés à utiliser :

```
@Service.path("/titles")
export default class ServiceTitles {

 @Methods.get("/")
 @Hal.halServiceMethod(({pageSize:"limite", pageIdx:"index"}))
 getAll() {
  return BdHelper.getTitles(db);
 }
}
```

Du coup pour le premier cas il vous suffira d'utiliser la requête suivante pour paginer votre service :
/titles?pageSize=2&pageIdx =14 (cela renverra deux titres à partir de la position 14).
Pour le second cas on aura la requête :
/titles?limite=2&index=14 (qui aboutira au même résultat).
Si vous voulez tester cette requête avec Httpie la requête sera la suivante :

```
http http://localhost:8080/titles pageSize==2 pageIdx==14
```

Nous obtenons exactement le même flux Json dans les cas. Voici le flux de réponse : code complet sur www.programmez.com
Veuillez noter l'ajout des liens first, last, next et prev dans la partie lien Hypermédia. Cela permettra à votre client de mieux s'y retrouver dans la liste. Si jamais un élément est ajouté dans la liste en cours de pagination, il sera pris automatiquement en compte par le framework si vos données côté serveur sont à jour.

CONCLUSION

Nous avons donc, grâce à threerest et les décorateurs, fourni une API plus flexible à l'aide des liens hypermédia et géré de façon simple et rapide les paginations hypermédia. L'idée principale étant que la partie hypermedia est une surcouche à votre service rest et doit être couplée au minimum avec le côté HAL. Il y a donc un minimum de cohésion entre threerest et votre service. En espérant que cela vous a donné envie de vous lancer le grand bain de l'hypermédia ! •

Bibliographie

Les sources de tous les exemples de l'article sont présentes sous https://github.com/lynchmaniac/bdtheque
Le projet est appelé bdtheque et regroupe un ensemble de sous-projets.
bdtheque_simple correspond à la partie "NodeJS chez les développeurs" ;
bdtheque_express correspond à la partie "Express et compagnie" ;
bdtheque_routes correspond à la partie "Tour de Gaule de notre API" ;
bdtheque_threerest correspond à la partie "Le domaine d'HATEOAS".
Le framework Threerest est disponible sous https://github.com/wmoulin/threerest
Et Joobster sous https://github.com/wmoulin/joobster
Une explication sur les décorateurs JS:
https://blog.hadrien.eu/2015/06/05/decorateurs-es7/
Le modèle de maturité de Richardson :
http://martinfowler.com/articles/richardsonMaturityModel.html
Le requêteur HTTPie : https://github.com/jkbrzt/httpie

Un langage que tout le monde **Elm** !

Jordane Grenat
VISEO

Ces dernières années, on a pu découvrir de nombreuses bonnes pratiques dans notre quête du développement Web. Approche orientée composants, programmation fonctionnelle, immutabilité, ajout de types, virtual DOM, architectures de données unidirectionnelles, … Toutes ces évolutions reposent pour la plupart sur des bibliothèques JavaScript, très utiles mais insuffisantes pour garantir que ces notions soient effectivement respectées.

Et si nous repensions aujourd'hui le développement Web pour créer un langage qui intègrerait, et surtout garantirait, ces bonnes pratiques. Ceci pour pouvoir construire des applications Web de la meilleure façon possible ? Ce langage, c'est Elm.

Présentation et syntaxe

Elm est un langage fonctionnel fortement typé qui compile en JavaScript. Il est actuellement surtout utilisé en front-end pour créer des interfaces, que ce soit un simple site Web, une application Web plus complexe (quiz, site d'e-learning…) ou même bien souvent pour des jeux dans le navigateur. La syntaxe de base, que nous allons découvrir maintenant, est plutôt semblable au JavaScript. Pour cela, commençons par télécharger et installer Elm en suivant les instructions présentes sur la page d'installation du site officiel (https://guide.elm-lang.org/install.html). Une fois installé, nous avons alors accès à plusieurs utilitaires en ligne de commande, dont elm-repl qui nous donne accès à un bac à sable en Elm. On peut alors tester quelques instructions basiques :

```
> elm-repl
---- elm-repl 0.18.0 -----------------------
:help for help, :exit to exit, [...]
---------------------------------------------
> "Hello"
"Hello" : String
> 3 + 4
7 : number
> 12.5
12.5 : Float
```

On remarque qu'à chaque instruction, le REPL nous affiche la valeur de retour et le type de données, qui commence toujours par une majuscule. Dans le cas de l'instruction 3 + 4, pourtant, on obtient un *number* avec une minuscule. En effet, le compilateur n'est pas capable de déterminer s'il s'agit d'un Int ou d'un Float – les deux types numériques en Elm. Ce n'est pas le cas avec 12.5 puisqu'il constate directement qu'il ne peut s'agir que d'un Float. On en sait déjà assez pour créer notre première fonction ! Ici, la syntaxe est beaucoup plus concise qu'en JavaScript puisqu'on évite les parenthèses, virgules, accolades et *return* superflus :

```
> sum a b = a + b
<function> : number -> number -> number
```

On déclare ici une fonction sum prenant deux paramètres – a et b – et retournant la somme de ces deux nombres. On voit que le type retourné est un type de fonction, annonçant que celle-ci prend deux nombres en paramètres puis retourne un nombre. Pour l'utiliser, on procède ainsi :

```
> sum 3 4 7
14 : number
```

On sait maintenant déclarer une fonction et l'utiliser, mais nous n'avons même pas encore parlé de variables ! Eh bien en Elm, une variable est une fonction sans paramètres :

```
> myNumber = 3.7
3.7 : Float
> myNumber * 2
7.4 : Float
```

Pour les conditions également, on peut voir que la syntaxe est plus concise qu'en JavaScript :

```
> if True then "Vrai" else "Faux"
"Vrai" : String
> if myNumber > 0 then "Positif" else "Négatif"
"Positif" : String
```

On peut également créer des tuples – des structures de données comportant un nombre fixe d'éléments :

```
> (1.3,"string")
(1.3,"string") : ( Float, String )
```

On a aussi des listes et des Dict – ou dictionnaires – qui sont assez semblables aux objets JavaScript :

```
> ["Element 1", "Element 2"]
["Element 1","Element 2"] : List String
> { key1 = "value1", key2 = 3 }
{ key1 = "value1", key2 = 3 } : { key1 : String, key2 : number }
```

Les types sont ici plus complexes. Le type List prend en effet en argument le type de ses éléments, et le type d'un Dict décrit précisément sa structure.

Un langage au compilateur humain

Avant d'aller plus loin, parlons d'une des forces de Elm : son **compilateur**, qui est sans doute le plus utile que j'aie pu rencontrer. Ses messages d'erreur sont limpides et très pédagogues : il nous explique l'erreur qu'il a rencontrée et donne des indices très précis pour résoudre le problème. Voyons un exemple concret en essayant de concaténer deux chaînes de caractères :

```
> "Hello " + "World!"
-- TYPE MISMATCH ----------------------------------------------
repl-temp-000.elm
The left argument of (+) is causing a type mismatch.
3|   "Hello " + "World!"
```

```
^^^^^^^^
is expecting the left argument to be a: number



But the left argument is:

String

Hint: To append strings in Elm, you need to use the (++) operator, not (+).

<http://package.elm-lang.org/packages/elm-lang/core/latest/Basics#++>
```

On a des indications claires : le compilateur s'attendait à un nombre et non pas à une chaîne de caractère, ce qui nous suggère que la concaténation ne se fait pas comme en JavaScript. Mieux encore, le compilateur *comprend* notre intention première et nous redirige vers l'opérateur ++ , documentation à l'appui, pour notre concaténation. Autant dire que durant votre apprentissage, ce sera votre meilleur professeur !

Un modèle de données très proche du métier

Le système de types est très performant et permet de déclarer des types alias et des union types à la volée. Le premier sert – comme son nom l'indique – à définir un alias pour un type, par exemple pour un Dict qu'on utilisera à plusieurs endroits :

```
> type alias User = { username : String, password : String }
> User "MyUsername" "MySecretPassword"
{ username = "MyUsername", password = "MySecretPassword" } : Repl.User
```

On voit qu'en créant un type, on génère en même temps un constructeur qu'on peut ensuite utiliser directement.
Les union types – quant à eux – sont bien plus utiles. Ils ressemblent à des *enum* en plus puissants, puisque chaque membre peut prendre zéro, un, ou plusieurs arguments. Imaginons que nous voulions charger une chaîne de caractères depuis un back-end. À tout instant, la valeur peut être déjà chargée, mais aussi potentiellement en cours de chargement, non chargée, ou en erreur. En JavaScript, on pourrait utiliser ce modèle :

```
const myString = null; const status = 'pending'; const onError = false;
```

En Elm, on peut utiliser l'*union type* suivant :

```
type RemoteString = NotLoaded | Loading | Loaded String | OnError Int String
-- Puis :
myString = Loading
-- Ou :
myString = Loaded "myString"
-- Ou encore :
myString = OnError 404 "Not Found"
```

Notre modèle de données correspond exactement à ce qu'on veut modéliser, et c'est là l'un des avantages de Elm ! Mais la façon dont Elm gère ces types rend possible le principal avantage du langage :

En Elm, il n'y a pas d'erreur au runtime

Vous pouvez relire ce titre. Et le relire encore. Par bien des façons, Elm garantit que vous ne rencontrerez jamais ces erreurs si présentes en JavaScript. Comment ? En vous forçant à prendre tous les cas en considération. Si un cas est possible, vous devez le gérer. Par conséquent, un if est **toujours** suivi d'un else. Quand on veut gérer les *union types*, Elm nous propose une syntaxe ressemblant à un switch en JavaScript :

```
status =
  case myString of NotLoaded -> "Test"
     NotLoaded -> "Not Loaded"
     Loaded value -> "String loaded: " ++ value OnError
     code status -> "Error: " ++ status
```

Avec cette syntaxe, vous êtes obligé de gérer toutes les branches possibles, ce qui vous assure de ne pas rencontrer de valeur inattendue au runtime. Mais la principale raison pour laquelle Elm n'a pas de *runtime errors* est l'absence de null ou d'undefined . En Elm, pour représenter une valeur qui peut être vide, on utilise Maybe qui n'est autre… qu'un *union type* ! Une valeur pouvant être non définie est donc représentée soit par un Nothing ou par un Just :

```
> Just "myValue"
Just "myValue" : Maybe.Maybe String
> Nothing
Nothing : Maybe.Maybe a
```

L'architecture Elm

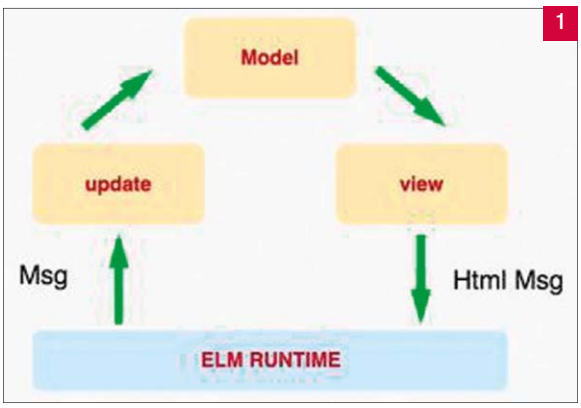
A force de réaliser des applications Web en Elm, une architecture commune a émergé et été standardisée sous le nom de *The Elm Architecture* (TEA). Il s'agit d'une architecture de données dans laquelle la donnée circule dans un seul sens à l'instar de Redux. Et pour cause, puisque le créateur de ce dernier s'est directement inspiré d'Elm ! [1]
Comme on peut le voir, on définit le modèle de notre application, et celui-ci est ensuite transmis à une fonction view qui génère le rendu. Cette vue peut ensuite déclencher des messages qui seront transmis à une fonction update avec notre modèle pour générer le nouveau modèle.
Prenons l'exemple – tiré de la documentation – d'un compteur. Il est possible d'incrémenter ou de décrémenter à loisir ce compteur. On commence par déclarer notre fichier comme étant un module nommé Main :

```
module Main exposing (..)
```

Notre modèle de données est donc un entier, qu'on initialise à 0 :

```
type alias Model = Int


model : Model
model = 0
```

On remarque à cette occasion qu'il est possible – et même fortement conseillé – d'ajouter des annotations à nos éléments pour indiquer leur



1

type. Cela permet d'avoir une double-vérification : le compilateur n'a pas besoin de ces indications et va déduire de son côté le type des données et les comparer. En plus d'ajouter de la sécurité, cela permet d'avoir une documentation rapide de notre code et de réfléchir clairement à notre API. Notre application peut recevoir deux types de messages, l'un pour décrémenter, l'autre pour incrémenter :

```
type Msg = Increment | Decrement
```

On a maintenant tous les éléments qu'il faut pour créer notre fonction de mise à jour du modèle. Celle-ci prend en paramètre un message – de type Msg – puis le modèle courant pour retourner un modèle modifié :

```
update : Msg -> Model -> Model
update msg model =
  case msg of
    Increment -> model + 1
    Decrement -> model - 1
```

Dans une application respectant l'architecture Elm, notre fonction d'update est généralement un case sur le message reçu. L'ajout d'une fonctionnalité consiste donc souvent à rajouter un nouveau membre à notre type Msg, puis à laisser le compilateur nous guider sur les modifications à effectuer dans le code ; une forme de *compilator-driven development* !
Nous avons maintenant un modèle, des messages, et une façon de mettre à jour ce modèle en fonction des messages. Il ne nous reste plus qu'à afficher la vue selon notre modèle.
En Elm, le HTML s'écrit… en Elm ! On a effectivement des fonctions contenues dans un module Html qui permettent de décrire une structure de données que le compilateur traduira ensuite en HTML. Chaque balise correspond donc à une fonction prenant généralement deux arguments. Le premier correspond à la liste des attributs – il y a des fonctions pour les attributs également – et le second est une liste d'enfants.
On pourrait ainsi afficher notre compteur comme ci-dessous. Vous pouvez noter au passage la syntaxe d'import au début du code, permettant d'inclure des éléments d'autres modules :

```
import Html exposing (Html, button, div, text) import
Html.Events exposing (onClick)

view : Model -> Html Msg
view model =
  div []
  [ button [ onClick Decrement ] [ text "-" ]
    div [] [ text (toString model) ]
    button [ onClick Increment ] [ text "+" ]
    ]
```

La syntaxe semble contraignante sur des exemples simples, mais on se rend très vite compte quand on monte en complexité qu'il s'agit d'un véritable avantage : tous les outils du langage sont à notre disposition pour nous faciliter la vie ! Et nous verrons plus loin que cela simplifie également les tests.
Nous avons donc nos quatre briques de base : un modèle, des messages, une fonction d'update, et une fonction de view. Il ne reste plus qu'à créer un programme en lui transmettant ces éléments !

```
main =
  Html.beginnerProgram { model = model, view = view, update = update }
```

Ici, on utilise pour cela Html.beginnerProgram, une version simplifiée d'un programme Elm. La version plus complexe permet de communiquer avec JavaScript – ce que nous allons voir un peu plus bas – et d'introduire des effets secondaires.
Une fois notre code compilé avec elm-make , un fichier *Main.js* est généré qu'on utilise ainsi dans notre fichier HTML :

```
<div id="root"></div>
<script src="./Main.js"></script> <script>
  const root = document.getElementById('root');
  Elm.Main.embed(root);
</script>
```

Les effets secondaires

Imaginons qu'on souhaite incrémenter notre compteur, non pas de 1 mais d'un nombre aléatoire compris entre 1 et 5. Comme tout est pur dans le langage, il n'est pas possible de générer des valeurs aléatoires ou de récupérer des données changeantes (par exemple via un appel REST). Tous ces effets secondaires sont en fait isolés et exécutés par le *Elm runtime*. Pour cela, nous allons introduire le concept de commandes. Une commande est une recette que l'on va transmettre à Elm pour qu'il l'exécute à notre place. Ce peut-être pour générer un nombre aléatoire, pour effectuer un appel REST, ou même par exemple pour mettre en place un timer. Pour commencer, définissons un vrai programme et non plus un beginnerProgram :

```
main =
  Html.program { init = init, view = view, update = update, subscriptions = (\_ ->
Sub.none) }
```

On remarque deux différences par rapport à notre précédent programme. On fournit une subscription , qui permet de s'abonner aux messages que peut envoyer le JavaScript. Ici, cela ne nous intéresse pas, aussi nous retournons toujours un Sub.none qui indique qu'on ne souhaite s'abonner à rien. La seconde différence est qu'au lieu de fournir un modèle, on fournit maintenant ce qui s'appelle init . Là aussi, il s'agit d'une valeur pour initialiser notre application, sauf qu'elle a une forme un peu particulière :

```
init : (Model, Cmd Msg) init =
(Model 0, Cmd.none)
```

Il s'agit d'un *tuple* contenant notre modèle et une commande – qui ici ne fait rien. Le Msg indique que nos commandes peuvent retourner des messages de type Msg . On va maintenant pouvoir transmettre des commandes au *Elm runtime* par ce biais. La fonction d'update renvoie désormais également un *tuple* :

```
update : Msg -> Model -> (Model, Cmd Msg)
update msg model =
  case msg of
    Increment -> (model + 1, Cmd.none)
    Decrement -> (model - 1, Cmd.none)
```

Pour générer un nombre aléatoire, on doit faire appel à un générateur de nombres grâce au module Random. Celui-ci contient une fonction int prenant en paramètres deux arguments : une valeur minimale et une valeur maximale.

```
import Random

randomValue : Random.Generator Int
randomValue = Random.int 1 5
```

On peut ensuite générer une commande à partir de ce générateur avec la fonction generate qui prend en arguments un constructeur de message acceptant en paramètre la valeur générée et notre générateur nouvellement créé. Voilà comment modifier notre application pour atteindre notre but :

```
type Msg = Increment | Decrement | IncrementWith Int |
DecrementWith Int

update : Msg -> Model -> (Model, Cmd Msg)
update msg model =
  case msg of
    Increment -> (model, Random.generate IncrementWith randomValue)
    Decrement -> (model, Random.generate DecrementWith randomValue)
    IncrementWith value -> (model + value, Cmd.none)
    DecrementWith value -> (model - value, Cmd.none)
```

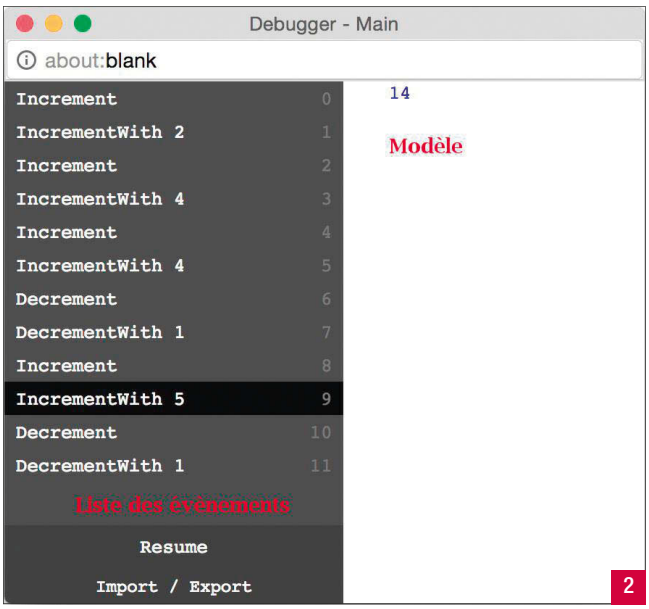
Cette façon d'isoler les effets secondaires peut paraître contraignante au départ, mais après de nombreux avantages. Comme tout est déterministe dans l'application, en enregistrant les évènements on peut reproduire exactement ce qui a produit un bug, et même revenir en arrière pour observer les modifications ! C'est d'ailleurs ce que permet l'outil *time travel*, présent lorsqu'on compile avec le *flag* --debug : [2]

Communication avec JavaScript

Ajoutons maintenant une fonctionnalité : dès qu'on quitte la page de notre compteur et qu'on revient sur la page, on souhaite que le compteur ait conservé la même valeur. Pour cela, nous allons utiliser le *localStorage*. En Elm, un module existe pour ce cas précis, mais agissons comme si nous l'ignorions, et passons par JavaScript pour le faire. Pour communiquer avec JavaScript, Elm utilise un système de *channels*, des flux de données appelés *ports* sur lesquels nous allons écouter des messages provenant de JavaScript, et d'autres sur lesquels nous allons pouvoir envoyer des messages. Pour commencer, on doit indiquer que notre module utilise des ports :

```
port module Main exposing (..)
```

Envoyer des messages est considéré comme un effet secondaire et c'est donc tout naturellement qu'on va utiliser une Cmd pour exécuter cette action. Pour cela, on déclare un *port*, grâce auquel on peut ensuite envoyer un message :




```
port saveCount : Int -> Cmd msg

-- On modifie ces deux branches dans notre fonction d'update :
IncrementWith value -> (model + value, saveCount (model + value))
DecrementWith value -> (model - value, saveCount (model - value))
```

Comme on peut le voir, une ligne suffit pour déclarer un *port*. On indique qu'on transmet un type *Int* et le tour est joué. Ensuite, on transmet une nouvelle commande chaque fois que le compteur est mis à jour. Côté JavaScript, on reçoit en retour de notre embed un objet contenant notamment sous la clé ports une liste des *ports* sur lesquels on peut écouter ou écrire :

```
const app = Elm.Main.embed(root);
app.ports.saveCount.subscribe(count => {
  window.localStorage.setItem('count', count.toString()); });
```

Pour émettre des messages, on va également définir un *port*, mais qui cette fois va retourner une Sub – ou *subscription* :

```
port initCount : (Int -> msg) -> Sub msg
```

On doit ensuite fournir un type de Msg dans lequel sera envoyé chaque message du JavaScript. On met également à jour notre définition du programme pour lui donner cette *subscription* :

```
main =
  Html.program
    { init = init
    , view = view
    , update = update
    , subscriptions = subscriptions
    }

type Msg
  = Increment
  | Decrement
  | IncrementWith Int |
  DecrementWith Int |
  InitCount Int

subscriptions : Model -> Sub Msg
subscriptions model = initCount InitCount
```

```
-- Et on rajoute cette branche à notre fonction d'update :
InitCount value -> ( value, Cmd.none )
```

La dernière chose qu'il nous reste à faire, c'est de publier un message côté JavaScript :

```
if (window.localStorage.getItem('count') !== null) {
  const count = parseInt(window.localStorage.getItem('count'), 10);
  app.ports.initCount.send(count);
}
```

Outillage et écosystème

Nous avons vu les principaux avantages de Elm en lui-même, mais il est toujours intéressant de se renseigner sur l'écosystème d'un langage avant de l'adopter.

Si le langage est encore un peu jeune (2012), sa communauté est très

dynamique et son environnement bien développé. Les montées de version se font maintenant presque sans heurts puisque grâce à la nature du langage, il est possible de livrer des utilitaires pour migrer une codebase automatiquement ! Et comme nous l'avons vu, si par hasard un module n'existe pas encore en Elm (ce qui devient de plus en plus rare), on peut communiquer avec du JavaScript pour réutiliser des bibliothèques existantes.

Les éditeurs et IDE les plus connus – comme Atom, Brackets, Emacs, IntelliJ, Sublime Text, Vim, VS Code – sont adaptés au développement avec Elm grâce à un simple plugin.

Pour ce qui est du déploiement, le système de build est installé automatiquement avec Elm, accessible via la commande elm-make . Le code est épuré et minifié. Un flag --debug permet d'ajouter l'outil de *time travel* très utile pour des sessions de débogage.

De nombreux tutoriels sont déjà disponibles sur Internet et les réponses sur le slack du langage sont très rapides. La communauté est prête à aider, et le créateur du langage consacre beaucoup de temps à celle-ci.

Tests

En Elm, il existe une bibliothèque standard pour les tests, elm-test . Le fait que les effets secondaires soient totalement isolés permet d'éviter les mocks et stubs qu'on retrouve dans la plupart des bibliothèques de tests. Chaque fonction est pure et peut ainsi être testée séparément. Voici comment nous pourrions tester notre application :

```
module Tests exposing (..)

import Test exposing (..)
import Expect
import App exposing (update, Msg(Increment, Decrement))

all : Test
all =
  describe "Test suite"
    [ test "Increment with 2" <|
      \() ->
        Expect.equal (update (IncrementWith 2) 7) (9, Cmd.none)
    , test "Decrement with 5" <|
      \() ->
        Expect.equal (update (DecrementWith 5) -3) (-8, Cmd.none)
    ]
```

Notre test consiste ici simplement à alimenter notre fonction d'update avec des messages et de tester le modèle retourné.

CONCLUSION

Quand on arrive de l'univers JavaScript, on se rend vite compte qu'Elm adresse de nombreux problèmes qu'on rencontre au quotidien. Toutes ces raisons – et bien d'autres encore – font que Elm est à mes yeux le langage le plus adapté au développement front-end actuel.

Il apporte aux développeurs des garanties et des sécurités très intéressantes à court et moyen termes, mais aussi surtout à long terme, puisque les réécritures de code peuvent s'envisager sans craindre de casser l'existant. L'adoption est encore faible mais augmente rapidement et des entreprises soutiennent activement le projet, à l'image de NoRedInk qui a d'ailleurs récemment recruté le créateur du langage pour lui permettre de travailler à plein temps dessus ! Il semble également que de grands acteurs comme Microsoft et Mozilla commencent à l'adopter sur des projets internes. Un signe qu'il est sans doute temps de s'y intéresser ?

Présentation du SharePoint Framework

• Sobitha JEYAKUMARAN,
Développeuse SharePoint chez
Infinite Square

• Gaëtan BOUVERET,
Tech Lead chez **Infinite Square et MVP Office Dev**
Blog : https://blogs.infinitesquare.com/

Voici déjà presque un an que la preview du SharePoint Framework a été publiée par Microsoft, suivie de la disponibilité générale (« GA – Generally Available ») en Février 2017 sur l'ensemble des tenants Office 365. Attendu comme une petite révolution par les développeurs SharePoint et fortement mis en avant par Microsoft, qu'est-ce que ce Framework et qu'apporte-t-il de nouveau ?

Pour commencer, malgré ce que son appellation semble indiquer, le « SharePoint Framework » (SPFx) n'est en réalité pas une nouvelle API mais plutôt une nouvelle expérience de développement : de nouveaux outils en phase avec les pratiques actuelles du Web, la possibilité de tester sans nécessiter SharePoint, ainsi qu'un déploiement simplifié. Son origine provient du besoin de donner un cadre à des pratiques existantes (injection de code JavaScript, déploiement de pages personnalisées) nées des contraintes de développements sur SharePoint Online qui ne supporte pas les solutions classiques serveurs, mutualisation et sécurisation obligent. A ce jour, le SPFx n'est disponible que sur SharePoint Online, c'est-à-dire au sein de l'offre Office 365, mais sera disponible d'ici la fin d'année sur SharePoint On-Premise au travers de la future mise à jour « Feature Pack 2 ».

Ce framework permet à ce jour de créer des WebParts (composants pouvant être disposés et personnalisés sur les pages par les concepteurs de site) à l'aide uniquement de JavaScript. On retrouve ainsi une structure de projet capable de s'appuyer sur une des principales librairies JS (jQuery, KnockoutJS et ReactJS) pour définir tout le fonctionnement : rendu, options personnalisables, ressources tierces... Cette orientation « Client Side » va dans le sens des dernières évolutions et tend à limiter les développements serveurs en exploitant l'API Cliente JavaScript et/ou REST, ce qui était déjà l'approche des add-ins SharePoint apparus avec la version 2013 (ces derniers restent d'ailleurs toujours d'actualité).

Nous allons voir tout cela avec la réalisation de notre premier projet utilisant le Framework SharePoint.

Présentation de l'environnement de développement

L'arrivée de SPFx a totalement changé notre façon de développer sur SharePoint. En effet, il est maintenant possible d'avoir un environnement SharePoint complet sur Windows, Mac ou Linux grâce à l'utilisation d'outils Open Source, gratuits, sur ces plateformes. Plus besoin de Windows Server ou de Visual Studio, les développeurs peuvent personnaliser SharePoint et le rendre « responsive » en utilisant les compétences du moment en développement Web.

Préparation de l'environnement SharePoint Framework avec Node.js

Différents outils sont nécessaires afin de monter un environnement SPFx. Nous allons dans cette partie faire un tour rapide de ceux-ci.

Node.js

Vous avez sûrement entendu parler de Node.js, la plateforme Open Source de programmation JavaScript. Eh bien la plupart des outils de développement JavaScript présents dans la toolchain de SPFx se basent sur cette plateforme. Pour l'installer rien de plus simple : il suffit de télécharger la dernière version LTS sur le site https://nodejs.org.

NPM

Npm est le gestionnaire de paquets officiel de Node.js. Il permet l'installation et l'administration de modules réutilisables et de dépendances. Pour installer npm, il suffit d'ouvrir une commande Node.js (en tant qu'administrateur) et de lancer la commande ***npm install-g*** (le –g permet l'installation globale).

Windows Build Tools

Windows Build Tools, également connu sous le nom de MSBuild ou Microsoft Build Engine est l'outil de compilation permettant la compilation de projets et solutions .NET sans avoir Visual Studio. Ils sont composés de Visual C++ Build Tools 2015 et de Python 2.7. L'installation se fait via PowerShell en tapant la commande ***npm install -g --production windows-build-tools***.

Installation du générateur de projet avec Yeoman

Afin de configurer simplement un projet de nouvelle Web Part avec SPFx, nous allons, dans un premier temps installer Yeoman.

Yeoman est un ensemble de trois outils :

- Yo : un outil CLI de scaffolding, générant donc la structure du projet tel que l'organisation des feuilles de styles, des scripts et des ressources.
- Un Build System : les plus connus sont Gulp et Grunt. Ce sont des gestionnaires de tâches (ou task runner) utilisés pour les tâches répétitives telles que l'automatisation de la minification, la compression...
- Un Package Manager : les plus connus sont NPM et Bower. Ce sont des gestionnaires de paquets permettant la gestion des dépendances.

Pour installer Yeoman (et Gulp, comme ça ce sera fait), nous allons lancer la commande ***npm install -g yo gulp***.

Nous allons maintenant utiliser un plugin Yeoman, generator-sharepoint pour initialiser notre nouveau projet de Web Part avec toutes les dépendances et configurations nécessaires. Il suffit pour cela de lancer la commande ***npm install –g @microsoft/generator-sharepoint***.

Langage et éditeur de code

Pour développer notre Web Part avec SPFx, nous allons utiliser TypeScript, un langage Open Source développé par Microsoft proposant une programmation orientée objet classique avec une notion de classe inexistante en JavaScript. C'est un langage compilé en JS.

Nous utilisons Visual Code, un éditeur développé par Microsoft, cross-platform et Open Source, mais au final sachez que n'importe quel éditeur de texte (notepad++, Sublime, …) fera l'affaire !

Debug grâce au Workbench local

Jusqu'à maintenant, travailler sur SharePoint signifiait déploiement d'une solution sur un serveur, ce qui pouvait prendre plus ou moins du temps en fonction du projet. Avec SPFx, nous avons le Workbench local, une page Web pour tester la Web Part sur son poste sans avoir besoin de se connecter à SharePoint. En résumé, il est maintenant possible de se passer de SharePoint, du moins tant que vous n'avez pas besoin de vous y connecter pour récupérer des données.

Place au code !

Nous allons créer notre première WebPart avec le SharePoint Framework.

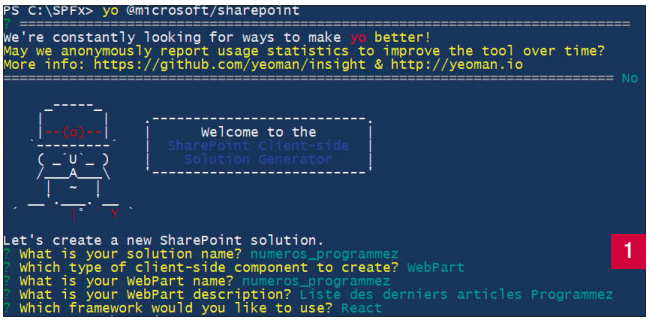
Création de la solution

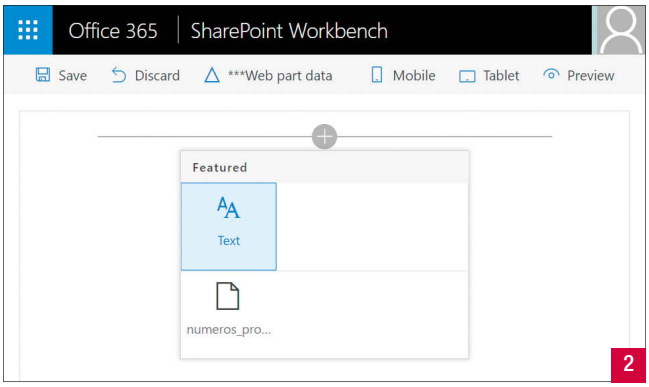
- Créer un dossier pour le projet numerosProgrammez ;
- Ouvrir une invite de commande et aller dans le dossier créé ;
- Générer un nouveau projet grâce à la commande ***yo @microsoft/sharepoint :***
  - Donnez un nom à la solution,
  - Choisissez le répertoire où sera enregistré la solution,
  - Donnez le nom de la WebPart et sa description,
  - Sélectionnez React comme Framework utilisé. [1]
- Remarque : la première fois, vous devrez exécuter la commande « ***gulp trust-dev-cert »*** afin d'installer le certificat développeur ;
- Exécutez ensuite « ***gulp serve »*** pour avoir un aperçu de votre site, en local, grâce au Workbench. [2]

Vous pouvez déjà ajouter la WebPart (qui s'appelle ici "numeros_programmez") qui pour l'instant ressemble par défaut à : [3]
Notez le petit menu contextuel sur fond noir pour la modifier ou la supprimer.

Modification du code client

Nous allons faire afficher à notre Web Part les cinq derniers numéros de Programmez. Pour éviter de nécessiter SharePoint, nous allons simuler l'existence des données dans un fichier TypeScript.





Ouvrez la solution dans VSCode (pour ceux qui ne connaissent pas, il suffit de faire File -> Open Folder et de choisir le répertoire) : comme vous pouvez le voir, nous avons déjà plusieurs répertoires dans la solution mais celui qui nous intéresse est "src" contenant le code et les styles de la Web Part. [4]

Simulation des données

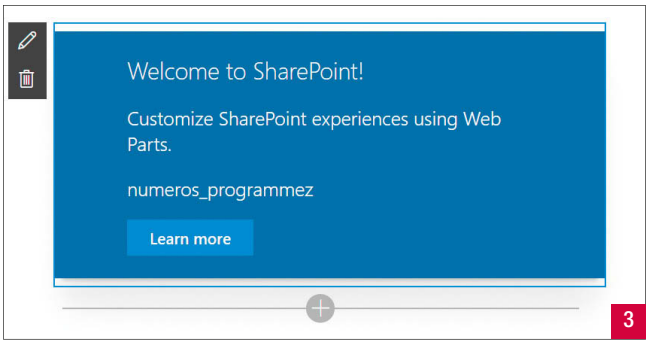
Pour simuler les données, créez un nouveau fichier TypeScript *MockHttpClient.ts* dans le répertoire **src/webparts/numerosProgrammez**.
Pour chaque numéro de Programmez, nous fournirons son numéro, son mois et année de publication ainsi qu'une image de sa couverture.
Nous allons donc créer une interface ISPList avec un numéro, un mois, une année, un titre et une image, comme ceci :

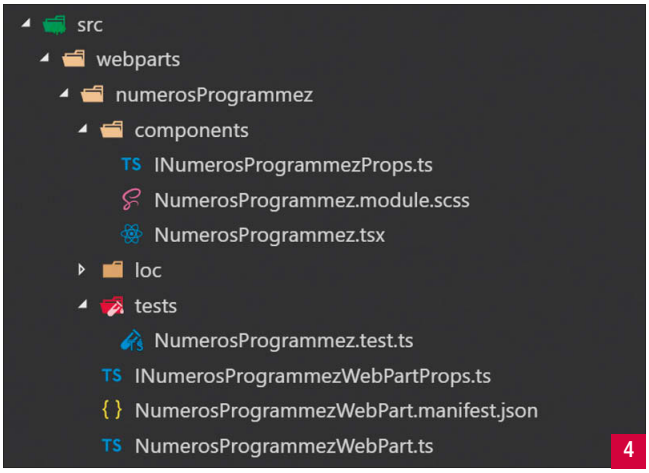
```
export interface ISPList {
Numero: number;
Mois: string;
Annee: number;
Title: string;
Picture: string;
}
```

Mais également une interface ISPLists :

```
export interface ISPLists {
value: ISPList[];
}
```

Le but de ce fichier *MockHttpClient.ts* est de simuler des données; ajoutons quelques entrées comme nous le ferions dans une liste SharePoint. Pensez bien à créer un répertoire dans le répertoire de la Web Part afin de stocker également les images (assets par exemple).





```
export default class MockHttpClient {
private static _items: ISPList[] =
[
{ Numero: 204, Mois: 'Février', Annee: 2017,Title: 'Le développeur est-il « écologie-compatible » ',
 Picture: '../src/webparts/numerosProgrammez/assets/couv-204.jpg' }, ...
]
}
```

Récupération des données

Maintenant que nous avons notre contenu, nous allons les afficher. Pour cela, modifiez le fichier principal de la WebPart, *NumerosProgrammez WebPart.ts*.

Avant de faire une quelconque modification, nous allons déjà ajouter les lignes suivantes en début de fichier afin d'importer les interfaces et données ainsi que le fichier de styles « scss » généré par Yeoman :

```
import styles from './components/NumerosProgrammez.module.scss';
import { ISPList } from './MockHttpClient';
import { ISPLists } from './MockHttpClient';
import MockHttpClient from './MockHttpClient';
```

Nous allons rajouter les deux lignes suivantes dans la méthode *render* qui s'occupe de la génération du rendu : on cible la div d'affichage, on appelle la méthode de récupération des données :

```
this.domElement.innerHTML = `<div id="numerosProgrammezFromMockList" />`;
this._renderListAsync();
```

Si vous regardez le code, vous pouvez voir que la fonction *renderListAsync* récupère les données et les affiche grâce à la fonction *renderList* :

```
private _renderListAsync(): void {
this._getMockListData().then((response) => {
this._renderList(response.value);
});
}
private _getMockListData(): Promise<ISPLists> {
return MockHttpClient.get(this.context.pageContext.web.absoluteUrl)
.then((data: ISPList[]) => {
var listData: ISPLists = { value: data };
return listData;
}) as Promise<ISPLists>;
}
```

RenderList s'occupe du rendu des éléments de la liste ISPList :

```
private _renderList(items: ISPList[]): void {
let html: string = '';
items.forEach((item: ISPList) => {
html += `
<div class="ms-Grid ${styles.customContainer}">
<div class="ms-Grid-row">
<div class="ms-Grid-col ms-u-sm12">
<div class="ms-Grid-col ms-u-sm12 ms-u-md6 ms-u-lg3 ">
<img class="${styles.picture}" src="${item.Picture}">
</div>
<div class="ms-Grid-col ms-u-sm12 ms-u-md6 ms-u-lg9 ">
<div class="ms-Grid-col ms-u-sm12 ms-fontColor-themePrimary">
<span>${item.Mois}</span> <span>${item.Annee}</span> <span>(Numéro ${item.Numero})
</span>
</div>
<div class="ms-Grid-col ms-u-sm12">
<span class="ms-fontColor-magentaDark ms-font-xl ms-fontWeight-semibold"> ${item.Title}
</span></br>
</div>
</div>
</div>
</div>
</div> `;
});
const listContainer: Element = this.domElement.querySelector('#numerosProgrammezFrom
MockList');
listContainer.innerHTML = html;
}
```

Vous l'avez peut-être remarqué, nous avons utilisé des classes proposées par Office UI Fabric, un Framework graphique similaire à Bootstrap pour les addins SharePoint et Office, créé par Microsoft et utilisé pour les interfaces de ses solutions Web telle que OneDrive. Ainsi nous aurons une WebPart facilement responsive. Vous pouvez également voir qu'il y a deux classes implémentées dans le fichier scss.

```
.customContainer{
background-color: #eee;
border-radius: 5px;
border: 1px solid #d0d0d0;
padding: 10px;
margin-bottom: 20px;
.picture{
width: 120px;
}
}
```

Affichage des données

Si vous avez attentivement observé le code, vous verrez qu'il fait également référence au composant React principal qui est le fichier *NumerosProgrammez.tsx*.

Ce fichier possède une méthode render qui va s'occuper de l'affichage de notre WebPart comme ceci :

```
public render(): React.ReactElement<INumerosProgrammezProps> {
return (
<div>
<div id="numerosProgrammezFromMockList"></div>
</div>
);
}
```

Relancez **gulp serve** si nécessaire et profitez de votre première Web Part responsive réalisée avec le SharePoint Framework ! [5]

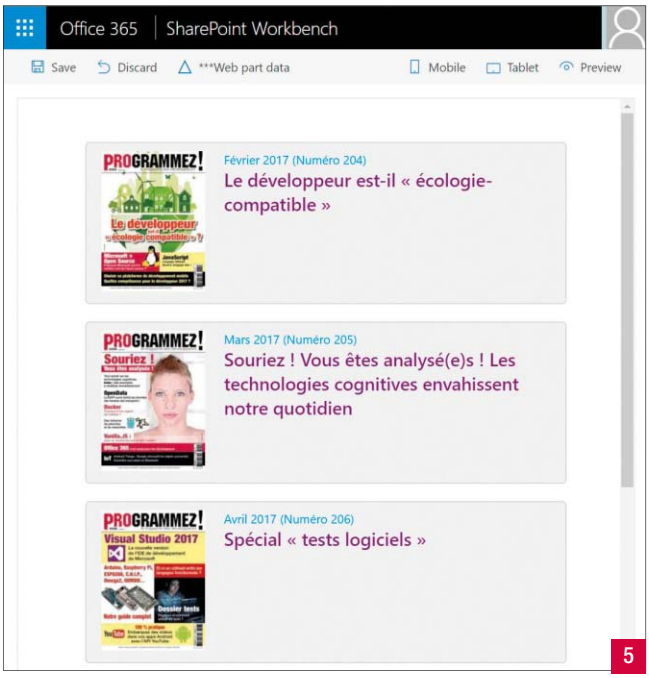
Déploiement de votre WebPart

Maintenant que la WebPart est développée et testée, il nous reste à la déployer pour être disponible pour tout le tenant Office 365. Nous avons 2 opérations à effectuer : d'un côté publier les ressources (JS/images/CSS) sur un CDN ou un espace partagé et de l'autre packager la WebPart pour qu'elle puisse être ajoutée par n'importe qui sur une page.

Pour la publication sur un CDN, il est possible d'utiliser directement Azure Blob Storage et un CDN associé. Il vous suffira, une fois ce dernier configuré, d'éditer le fichier « **deploy-azure-storage.json** » situé dans le répertoire « config » pour y spécifier le compte, le nom du container et la clé d'accès à utiliser :

```
{
  "workingDir": "./temp/deploy/"
  "account": "moncompteblobstorage",
  "container": "mywebpart",
  "accessKey": "q1UsGWocj+ j41avlG73rynbvKizZpIKK9XpnpA=="
}
```

Editez ensuite le fichier « **write-manifests.json** » pour donner le chemin de votre CDN, par exemple :



5

```
{
  "cdnBasePath": "https://programmez.azureedge.net/mywebpart/"
}
```

Vous pouvez désormais exécuter la commande « **gulp --ship** » pour générer tous les assets dans le répertoire « temp\deploy » (celui qui correspond à la variable « workingDir » du fichier de déploiement vu précédemment, suivie d'un « **gulp deploy-azure-storage** » pour un aller direct dans les nuages.

Pour finir, la commande « **gulp package-solution** » vous permet de générer un fichier « .sppkg » que vous pourrez déposer dans le catalogue d'App. Pensez à ajouter le paramètre « **--ship** » pour que ce package contienne la référence à vos fichiers déployés dans le CDN car sinon il continuera à pointer vers vos fichiers en localhost; pratique pour les tests, moins pour la production J.

Enfin, sachez que si vous ne voulez pas forcément utiliser le CDN d'Azure (ou autre), Microsoft propose d'associer une de vos bibliothèques SharePoint à un CDN. C'est gratuit, il vous faudra utiliser PowerShell pour activer cette fonctionnalité et la paramétrer. Attention cependant, les fichiers mis à disposition pour le CDN sont accessibles publiquement aussi, même si l'URL reste relativement complexe et difficile à deviner, évitez d'y mettre du contenu sensible.

Pour en savoir plus

Voilà, vous êtes maintenant capable de créer votre première WebPart avec le SharePoint Framework, la tester et la déployer sur votre environnement ! Comme vous avez pu le constater, SharePoint a pris le virage du Web moderne, tant côté outillage que côté code avec l'adoption des librairies JavaScript telles que ReactJS, et ne nécessite plus l'installation d'une machine virtuelle pour le moindre développement.

Pour aller plus loin, une seule adresse : http://dev.office.com. Vous y trouverez la documentation, le guide pas à pas de l'installation de votre environnement de développement, des exemples, ainsi que les nouveautés en preview. N'hésitez pas non plus à suivre de près l'actualité de la communauté du groupe SharePoint PnP (Patterns and Practices) : http://aka.ms/SharePointPnP.

A vous de coder !

Réalisez votre première Instant App !

• Nicolas Telera
OCTO Technology

Vous êtes un adepte des Android App Links(1) et vous trouvez formidable d'avoir la possibilité de suggérer à vos utilisateurs d'installer votre application ou de les rediriger vers celle-ci depuis votre site, et cela avec les données spécifiques à leurs requêtes ? Et bien oubliez (presque) tout ce que vous savez.

Depuis la Google I/O 2017, Google a rendu public pour les développeurs ce qui se présente comme la plus grosse nouveauté dans le monde du développement Android : les Instant Apps.
Les Instant Apps sont des applications qui peuvent être exécutées instantanément, à partir de n'importe où, sans avoir besoin d'être installées sur votre smartphone. Vous pouvez ainsi afficher une carte, passer une commande ou encore afficher un fil d'actualités à travers votre application, sans l'installer, à partir d'un simple tap sur un lien provenant d'un site internet ou même d'un simple SMS. Cet article présentera une vue d'ensemble des Instant Apps afin de vous permettre de commencer à l'utiliser dans votre propre développement. Nous n'entrerons pas dans les moindres détails dans la mesure où le site des développeurs Android(2) le fait déjà très bien. Et la bonne nouvelle est que vous pourrez l'utiliser dans votre application existante sans avoir à la refaire à zéro !

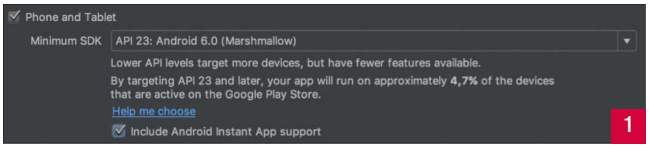
Ce dont vous aurez besoin

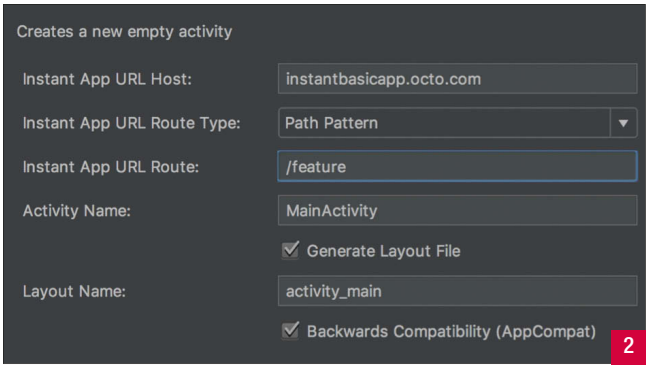
Afin de pouvoir commencer à développer des Instant Apps, vous devrez installer la version Canary 1 d'Android Studio 3.0(3) ainsi que le Instant Apps SDK (disponible par le SDK Manager). Vous devrez également configurer l'API minimum de votre projet à 23 (Android 6.0), le support pour les API 21 (Android 5.0) sera disponible prochainement. Enfin, il faut savoir que les Android App Links sont utilisés dans les Instant Apps, mais vous n'avez pas forcément besoin d'y être familiarisés pour pouvoir commencer à construire votre première app.

Comment ça fonctionne ?

Comprendre le fonctionnement des Instant Apps est relativement simple et peut être résumé en 3 grandes parties :

1 – Imaginez votre application non pas comme un tout, philosophiquement séparé en fonctionnalités mais comme des fonctionnalités indépendantes, regroupées sous une base commune (oui, il y a une différence).





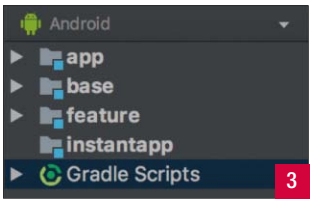
2 – Maintenant, nettoyez tout ça et ne conservez que le minimum vital à toutes les fonctionnalités de votre application dans cette même base commune, puis répartissez le reste entre ces fonctionnalités.

3 – Enfin, associez chacune des fonctionnalités à une URL unique et laissez Google Play s'occuper du reste.

Google Play analysera la requête, vérifiera s'il existe une Instant App correspondant à l'URL et enverra le code de la base commune (base feature APK) et le code de la fonctionnalité demandée (feature APK) à l'utilisateur. Si l'URL ne correspond à aucune application, le système Android sera alerté et un intent sera broadcasté.
Ainsi, lorsqu'un utilisateur désire utiliser la fonctionnalité n°3 de votre application, il ne recevra que le code nécessaire à l'exécution de la fonctionnalité n°3, qui sera supprimé par le système après son utilisation. Supposons que l'application installable comporte 10 fonctionnalités, on devine facilement le gain en performance et donc en expérience utilisateur, lorsque ce même utilisateur ne recevra que le code nécessaire lui permettant d'utiliser uniquement une seule de ces 10 fonctionnalités.
Aussi, une fois que l'utilisateur aura reçu le code d'une fonctionnalité et de la base commune, si cet utilisateur désire utiliser une autre fonctionnalité, il n'aura qu'à télécharger le code de cette autre fonctionnalité sans avoir à télécharger le code de la base commune à nouveau. Gain en performance et en expérience utilisateur à nouveau.
Les Instant Apps sont sécurisées et ne peuvent obtenir les mêmes capacités que les application installables (caméra, position…) que par l'utilisation de la Permissions at Run Time API(4). Pour ces mêmes raisons de sécurité, certaines capacités(5) ne sont pas disponibles avec les Instant Apps comme l'accès au stockage interne de l'appareil.
Ça y est ! Vous savez à présent comment fonctionnent les Instant Apps. Cela vous plaît ? Dans ce cas continuons.

Comment l'utiliser ?

Afin de réaliser votre première Instant App, commençons par créer un nouveau projet dans Android Studio 3.0. Vous pouvez désormais inclure le support pour les Instant Apps ce qui vous permettra de configurer la route de votre fonctionnalité principale. [1] et [2]



Ainsi, lorsqu'un utilisateur tapera sur le lien https://instantbasicapp.octo.com/feature, Google Play n'enverra que le code lui permettant d'exécuter la fonctionnalité « feature » de votre application. Vous obtenez alors 4 modules : [3]

(1) https://developer.android.com/training/app-links/index.html
(2) https://developer.android.com/topic/instant-apps/index.html
(3) https://developer.android.com/studio/preview/index.html
(4) https://developer.android.com/training/permissions/requesting.html
(5) https://developer.android.com/topic/instant-apps/prepare.html#restricted

Le module app :

Ce module utilise toutes les fonctionnalités de l'application afin de construire l'APK de l'application installée.
Son fichier build.gradle applique le plugin com.android.application et dépend du module de base et de chaque module de feature nécessaire à la construction de l'APK installable (dans ce cas : module et feature).

```
app/build.gradle

apply plugin: 'com.android.application'

android {
  //...
}

dependencies {
  implementation project(':base')
  implementation project(':feature')
}
```

Le module instantapp :

Ce module a la même responsabilité que le module app, mais pour l'Instant App. Il est vide et n'est nécessaire que pour la construction de l'Instant App APK avec toutes les fonctionnalités de l'application.
Son fichier build.gradle applique le plugin com.android.instantapp et dépend de tous les modules utilisés pour la construction de la feature APK (dans ce cas : base et feature).

```
instantapp/build.gradle

apply plugin: 'com.android.instantapp'

dependencies {
  implementation project(':base')
  implementation project(':feature')
}
```

Le module base :

Chaque module de fonctionnalité dépend du module base. Il ne contient que les ressources partagées et construit la base feature APK pour l'Instant App. Cependant, tout le code de l'application peut être contenu dans ce module si l'application ne contient qu'une seule fonctionnalité.
Son fichier build.gradle applique le plugin com.android.feature et référence toutes les fonctionnalités de l'application. Il spécifie également qu'il est la fonctionnalité de base avec la propriété baseFeature.

```
base/build.gradle

apply plugin: 'com.android.feature'

android {
  //...
  baseFeature true
  //...
}

dependencies {
  feature project(':feature')
  //...
}
```

Le module feature :

Chacune des fonctionnalités de l'application est définie dans un module spécifique. Ce module contient toutes ses propres ressources, rien de plus, et construit une feature APK pour l'Instant App et un AAR pour l'application installable. Il doit contenir au moins un point d'entrée sous la forme d'une activity, qui sera l'élément principal affiché à l'utilisateur.
Son fichier build.gradle applique également le plugin com.android.feature et spécifie sa dépendance au module de la base commune.

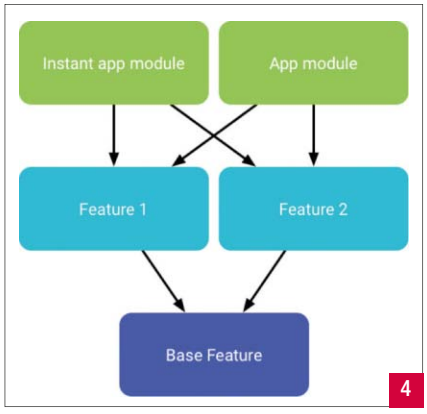
```
feature/build.gradle

apply plugin: 'com.android.feature'

android {
  //...
}

dependencies {
  implementation project(':base')
  //...
}
```

Ce schéma [4], tiré de la documentation de Google, représente l'architecture présentée ci-dessus. On peut y voir le module instant app ainsi que le module de l'application installable qui sont dépendants de chaque module de fonctionnalité, eux-mêmes dépendants du module de base commune.

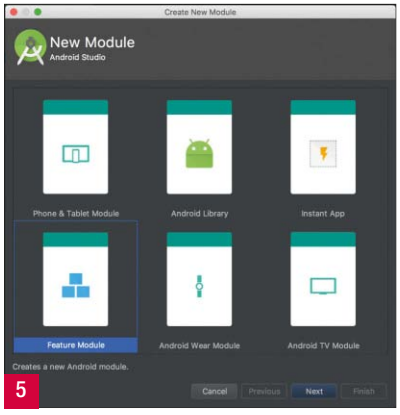
Plus de fonctionnalités ?

Dans ce qui suit, l'architecture de notre application a été mise à jour : « feature » devient « feature1 » et une « feature2 » a été ajoutée.

Vous pouvez désormais ajouter autant de fonctionnalités que vous voulez, chacune contenant ses propres ressources (couleurs, strings, drawables…), et son propre point d'entrée. Ajouter une nouvelle fonctionnalité à votre Instant App est très simple, Android Studio 3.0 offrant maintenant deux nouveaux types de modules : Instant App et Feature Module. Sélectionnez Feature Module pour ajouter une nouvelle fonctionnalité à votre application. [5]
Notez que chaque fonctionnalité est construite en tant que librairie et devra par conséquent être déclarée avec



4



5

un nom de package différent de celui de l'application. Au sein de chaque fonctionnalité, vous devrez ajouter certaines entrées spécifiques au fichier AndroidManifest.xml. Ainsi, vous ajouterez l'URL associée avec un *host*, un *pathPattern* et un *scheme*. Le point d'entrée de votre application nécessitera un <intent-filter> avec les attributs android.intent.category.LAUNCHER et android.intent.action.MAIN afin de permettre à Google Play de découvrir votre app. Dans ce même manifest de point d'entrée, vous devrez également définir l'URL par défaut de l'application par l'ajout d'un élément <meta-data> fournissant une HTTPS URL valide.
Il est également nécessaire d'ajouter un niveau de priorité au matching des URLs de vos fonctionnalités afin de permettre à Google Play de déterminer quelle fonctionnalité sélectionner. Par exemple, si votre application offre deux fonctionnalités basées sur une même URL pouvant être complétée avec un ID, un niveau de priorité différent sera attribué de la manière suivante :
https://instantbasicapp.octo.com/feature aura une priorité de 1
https://instantbasicapp.octo.com/feature/<ID> aura une priorité de 100

feature1/src/main/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  package="com.octo.instantbasicapp.feature1"
  xmlns:android="http://schemas.android.com/apk/res/android">

  <application>
    <activity android:name=".FeatureOneActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>

      <meta-data
        android:name="default-url"
        android:value="https://instantbasicapp.octo.com/feature1"/>

      <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>

        <data android:host="instantbasicapp.octo.com"/>
        <data android:pathPattern="/feature1"/>
        <data android:scheme="https"/>
        <data android:scheme="http"/>
      </intent-filter>

    </activity>
  </application>

</manifest>
```

Une fois que votre deuxième fonctionnalité est prête, vous voudrez probablement pouvoir y naviguer à partir de la précédente. Comment ? Et bien, chaque fonctionnalité peut être indépendamment accédée à partir d'une URL unique donc… Oui, vous avez deviné. Lors de l'utilisation d'une Instant App, vous pouvez naviguer à travers les fonctionnalités par l'utilisation d'intents configurées avec l'action Intent.ACTION_VIEW et une Uri.

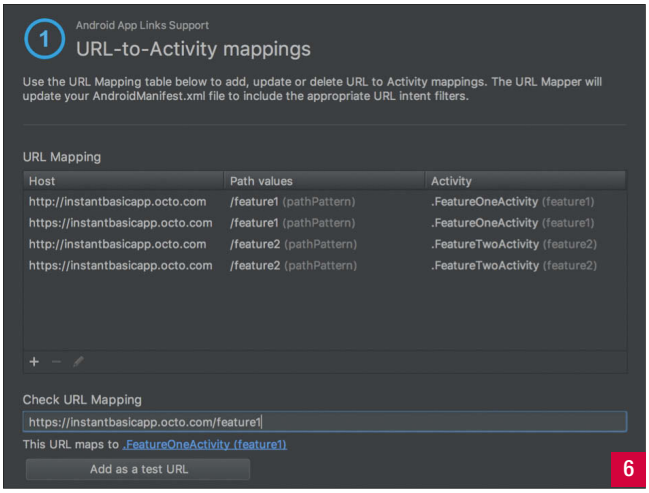
Maintenant que votre application est prête, sélectionnez le module instantapp dans le menu déroulant de sélection de la cible à exécuter, puis lancez-le sur votre appareil ou votre émulateur. Vous avez à présent votre fonctionnalité en cours d'exécution sur votre appareil, mais si vous vérifiez votre launcher, vous verrez que l'application n'a pas été installée !

Gérez vos URLs

Android Studio 3.0 offre également un assistant vous permettant de tester votre matching d'URLs. Sélectionnez Tools -> App Links Assistant pour afficher cet outil. Il vous permettra de gérer vos URLs en vérifiant localement votre matching, d'ajouter de la logique à vos activités mappées à des URLs vous permettant de transmettre des données à celles-ci, d'associer votre application à votre site Internet à travers les Digital Asset Links ou d'exécuter des URL entrées manuellement directement sur votre appareil ou votre émulateur. [6]
Notez que si la propriété de domaine de votre URL n'est pas vérifiée, vous ne pourrez pas publier votre Instant App. Voilà ! Votre première Instant App est prête et vous avez à présent tous les outils en main pour commencer à les utiliser dans votre développement.

Vous en voulez plus ?

Cet article est principalement basé sur la documentation Google des Android Instant Apps(6). Dans cette documentation, vous trouverez de nombreuses informations sur les bonnes pratiques liées au développement des Instant Apps, la gestion des permissions, l'identification ou encore l'In-app Billing. Il y a également un très bonne section FAQ contenant beaucoup de questions intéressantes telles que « Comment déployer mon Instant App sur Google Play ?(7) ».
Je recommande également très fortement la vidéo « Building an Android Instant App (Google I/O '17)(8) » qui présente un live-coding du refactoring d'une application Android existante vers une Instant App et ça vaut le détour !
Enfin, vous trouverez sur le Github des samples de Google (9) de nombreux exemples de code démontrant des Instant Apps avec une seule et plusieurs fonctionnalités. •
Enjoy!



6

(6) https://developer.android.com/topic/instant-apps/index.html
(7) https://support.google.com/googleplay/android-developer/answer/7381861
(8) https://www.youtube.com/watch?v=9Jg1D07Ngel
(9) https://github.com/googlesamples/android-instant-apps

L'informatique du programme Apollo est allée sur la Lune

• Yves Grandmontagne

Le programme spatial américain Apollo a marqué de son empreinte l'informatique naissante. On lui doit en particulier de grandes avancées dans le calcul, le premier ordinateur portable, la première femme programmeuse de renom. Et surtout le premier homme sur la Lune, avec la mission Apollo 11, le 20 juillet 1969 !

Neil Armstrong
© Nasa

Dans une période politiquement troublée par ce que l'on appelle 'la guerre froide', qui a opposé le bloc occidental regroupé derrière les Etats-Unis, au bloc communiste emmené par l'URSS, un combat unique va opposer les deux grandes puissances, la conquête de l'espace. Un temps gagné par les soviétiques - ils sont les premiers à avoir placé un satellite en orbite, Spoutnik, puis une capsule habitée par un animal, la chienne Laïka, et enfin un homme, Youri Gagarine - l'espace est devenu la hantise des gouvernants américains. C'est alors que le jeune et médiatique Président John Fitzgerald Kennedy va lancer, en 1961, un défi à l'Amérique : être les premiers à conquérir la Lune. Ainsi, il n'aura fallu que deux décennies pour que l'homme vainque l'attraction terrestre, rejoigne le vide sidéral, et pause le pied sur la Lune... Rien ne résiste aux pouvoirs de l'argent associé à la politique, souvenons-nous en ! Et pourtant, évoquer l'informatique du programme Apollo n'est pas très compliqué (!), à la condition de se rappeler le pouvoir et la volonté des hommes. C'est ce que nous allons évoquer dans cet article. En commençant, une fois n'est pas coutume, par vous relater une expérience personnelle... Ma rencontre avec Niel Armstrong, le premier homme qui a posé le pied sur la Lune !


© Nasa
Saturn V au décollage. Mission Apollo 11

Neil Armstrong l'ingénieur

C'était il y a quelques années, la date exacte n'a pas d'intérêt, quelques années également avant son décès. Invité à une conférence technique qui s'est tenue à Orlando (Floride), j'apprends, c'est très américain, que la clôture de la conférence sera assurée par Neil Armstrong. Passionné d'espace durant mon enfance - j'ai eu en particulier la chance d'assister en direct à la télévision à l'évènement du 19 juillet 1969 - l'homme était mon idole, et le rencontrer était un rêve. Rêve a demi accompli, puisque Neil Armstrong, qui vivait dans la paranoïa de découvrir un écrit signé de sa main proposé aux enchères sur eBay, refusait tout contact physique. Interdiction également de lui poser des questions. Tout juste ai-je reçu de sa main un exemplaire de sa biographie, sans autographe. Peu importe, l'homme est une figure historique de l'humanité, et durant plus d'une heure trente son discours nous a captivés. Qu'en retiendrons-nous ?

Le principal dans le discours de Neil Armstrong n'est pas l'aventure humaine, mais une aventure d'une catégorie d'humains, les ingénieurs ! Durant les programmes Mercury et Gemini, puis les débuts d'Apollo, la NASA n'a tenté d'expédier dans l'espace que des militaires. La raison en est multiple : d'abord la NASA est une émanation de la défense américaine, qui a bénéficié des budgets de la Darpa, la célèbre agence américaine qui finance les

LA NASA CRÉE LE MÉTIER DE PROGRAMMEUR

Le développement du logiciel qui intègre l'ordinateur AGC du module de commande et du module lunaire du programme Apollo a été stratégique pour la NASA qui a confié au MIT l'exercice délicat de créer le premier logiciel de l'histoire. Un demi siècle plus tard, nous retiendrons non pas le programme développé pour la NASA, mais la création d'une discipline aujourd'hui essentielle et incontournable, l'ingénierie logicielle, et d'un métier, celui de programmeur. Sans oublier, également, l'expérience acquise, par les universités et par les industries américaines, qui a fait des Etats-Unis la première nation de l'industrie informatique et de l'Internet.

projets militaires (la première ossature de l'Internet, par exemple, est issue d'un programme de réseau mené par la Darpa). Or, les investissements militaires, aujourd'hui encore, sont des moteurs de la recherche américaine. Les pilotes militaires, au delà de savoir piloter des avions, présentaient également l'avantage d'être une perte acceptable en cas d'échec. Neil Armstrong sera pourtant la première exception. Pilote de la Navy, il est recruté pour rejoindre le prestigieux mais dangereux programme Apollo. Il démissionne rapidement de la Navy, mais va rester dans le programme, en tant qu'ingénieur. Car au delà de l'aventure menée par quelques têtes brulées, comme nous les présentent Hollywood et la légende, le programme Apollo est une expérience technologique menée par des ingénieurs. Cela est important, nous y reviendrons dans l'évocation de l'informatique d'Apollo.

Ingénieur, Neil Armstrong l'est et le restera jusqu'au bout de sa vie. Aventurier ? Non, le programme Apollo est un projet d'ingénieurs, mené par des ingénieurs. Et c'est pour cela qu'il va conserver sa place, qui le mènera jusque sur la Lune. Non sans risque... Il nous révèlera qu'équipé de sa règle à calcul, il a évalué le risque encouru à prendre place dans une capsule exiguë placée en haut d'une bombe fusée de 3 000 tonnes : la probabilité que la mission échoue a été évaluée à 9 sur 10 ! Ce qui ne l'a pas empêché de « *poser son cul dans la capsule* »...

Cette anecdote nous permet d'introduire le premier ordinateur des ingénieurs de la NASA : la règle à calcul. Cet instrument bizarre, dont l'usage était encore enseigné dans nos écoles il n'y a pas si longtemps, permet de réaliser des calculs complexes avec une grande précision. Rappelons tout d'abord que les machines à calculer ne feront leur apparition qu'après la fin du programme Apollo, créées (tiens !) par une

Margaret Hamilton

L'unité AGC

IBM 360

entreprise d'ingénieurs dont les fondateurs s'appellent Hewlett et Packard.

N'allez pas croire, cependant, que les machines à calculer n'existaient pas à l'époque... De la taille d'une très grosse machine à écrire placée sur un meuble, on les appelait généralement facturières. Elles disposaient de quelques registres pour stocker temporairement les données calculées, et ainsi réaliser une facture ou une fiche de paie qui tenait sur une petite bande de papier. Mais les calculs réalisés étaient simples, très loin de ceux menés par les ingénieurs de la NASA, qui décidément préféraient leurs règles à calcul !

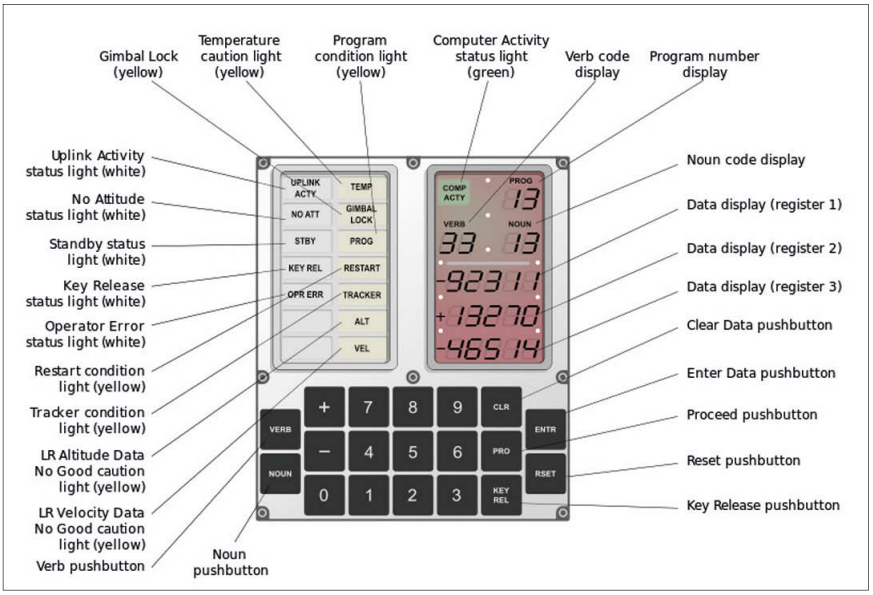
Le mainframe IBM 360

Et l'informatique dans tout cela ? Eh bien elle n'est pas loin! Nous avons évoqué la création d'une certaine entreprise Hewlett-Packard ; évoquons maintenant celle d'une autre entreprise d'ingénieurs, plus ancienne encore, une certaine International Business Machines. Eh oui, il s'agit bien d'IBM! L'administration spatiale américaine s'est dotée à partir de 1965 de 5 des ordinateurs les plus puissants de l'époque, des unités centrales IBM 360. Il s'agit de systèmes de calcul que la NASA réserve à la balistique. Car qu'est donc à l'époque un programme spatial, de fusées qui plus est, que d'expédier un objet dans l'espace et de le faire revenir ?

Successeur du deuxième calculateur accessible à un usage civil, l'IBM 360 (le premier est le 350), le modèle 360/75, entièrement câblé, offre une mémoire maximale de 1.048.576 octets (1 Mo de caractères), qu'il est possible de diviser en 4 blocs pour augmenter la performance, et qui opère à 750 nano secondes. La NASA l'a retenu également sur l'engagement d'IBM sur un fonctionnement de 40 000 heures sans erreur. Pour un prix de 3,5 millions de dollars en configuration haute, une fortune pour l'époque. Pour se rendre compte d'une part de la puissance du système, et de l'évolution de l'informatique depuis, sachez que selon Neil Armstrong, « *La puissance du temps de calcul nécessité par une recherche Google à l'heure actuelle correspond à celle nécessaire à l'ensemble du programme spatial Apollo, qui a duré 11 ans et a lancé 17 missions* ». Notons que l'architecture IBM 360/370, étendue depuis, est toujours utilisée sur la série Z d'IBM (systèmes zOS et Linux/390).

Margaret Hamilton entre en scène

Le 22 novembre 2016, Barack Obama, pour quelques jours encore président des Etats-Unis, a décoré de la *Presidential medal of Freedom*, la médaille présidentielle de la Liberté, et plus haute distinction civile américaine, Margaret Hamilton. Celle que la Maison Blanche a tenu à saluer, aussi appelée « *Mère du Logiciel* », s'est illustrée à une époque où les femmes étaient rares dans le monde de l'informatique. Elle n'a que 24 ans lorsqu'elle intègre le MIT (Massachusetts Institute of Technology), en 1960, comme programmeuse. La Nasa, investie en 1961 du projet Apollo, s'est rapidement tournée vers les universités américaines pour l'accompagner. C'est ainsi que Margaret Hamilton est invitée à travailler avec la NASA sur le logiciel informatique du programme Apollo. Elle rejoint l'équipe en charge de coder le premier ordinateur portable qui intégrera les capsules et le module lunaire pour gérer le décollage et l'alunissage. Margaret Hamilton va ainsi coder le tout premier logiciel de l'Histoire ! A l'époque, coder se traduit par des trous dans des cartes perforées. Pour créer le premier logiciel, elle va participer à la création d'un langage qui va marquer l'informatique, et qui aujourd'hui encore est le moteur de nombreux mainframes : COBOL. C'est d'ailleurs son autre surnom, la « Mère de Cobol ». Pour l'anecdote, le 20 juillet 1969, quand le LM, le module lunaire d'Apollo 11 s'apprête à se poser sur la Lune, l'ordinateur envoie des messages d'erreur. Mais il a été codé pour traiter en priorité les requêtes urgentes. Et il ne bogue pas ! Ce jour là, l'alunissage est un succès, et Neil Armstrong va être le premier homme à poser le pied sur la Lune !



AGC - Apollo Guidance Computer

Dans l'espace, la CM, capsule 'Command Module' qui emporte les trois astronautes durant la plus grande partie de la mission, et le LM, le module lunaire 'Lunar Module', ont emporté le même ordinateur, un AGC (*Apollo Guidance Computer*). Ses concepteurs l'ont surnommé « le quatrième membre d'équipage ». L'ordinateur est le plus compact de l'époque, il est même considéré comme le premier ordinateur portable. Sa taille est à peu près celle d'une feuille A3 d'une épaisseur de 5 centimètres, pour un poids de 32 kilos. Sa consommation électrique est de 70 watts à 28 volts. Il embarque 24 modules de circuits imprimés, chacun équipé d'un connecteur 72 pins, qui tournent à 1,43 Mhz. A l'époque le processeur n'a pas encore été inventé, et il est le premier à utiliser des circuits imprimés... Le premier modèle, conçu en 1961, embarquait 4 ko de mémoire fixe et une mémoire effaçable de 256 mots (!). Dès juin 1963, la capacité est portée à 10 Ko fixes et 1 Ko effaçable. Puis 32 Ko. C'est la NASA qui est responsable du ridicule petit volume mémoire, car elle avait tout simplement oublié d'indiquer sa taille dans ses spécifications ! AGC était scellé. Il était accompagné d'un écran et d'un clavier de chiffres exclusivement nommé DSKY (prononcez « disky »). Deux DSKY embarquaient dans le module (un pour le LEM), le premier sur la console de commande du tableau de bord principal (Main Display Console), relié à l'unité de mesure inertielle ; et le second placé à proximité de la station de navigation équipée d'instruments optiques, auxquels il était également relié.

Le MIT, qui a conçu l'AGC - il a été chargé de concevoir une machine pour aider à calculer les trajectoires et prendre parfois les commandes -, a retenu une longueur de données et d'instructions de 6 bits, alors que déjà à l'époque nombre de scientifiques étaient passés aux 24 bits. L'avantage de ce choix provient d'expressions de taille plus réduite, donc plus rapides, et une plus grande précision par l'usage de 'mots' multiples. Le premier niveau de précision était atteint par l'association de 2 mots, soit 14 bits. Les deux bits restants étant consacrés au chiffre '-' (moins) et à la parité. Une instruction était exprimée en 8 bits, les données vectorielles tri-dimensionnelles demandaient 3 mots pour s'exprimer. L'adressage direct de la mémoire était limité, un schéma de 'bank register' rendait possible l'adressage de toute la mémoire. Revers de la médaille, la simplification de cet équipement s'est traduite par une augmentation de la complexité du logiciel et de son développement. Pas de langage évolué à l'époque, les programmeurs devront se contenter d'un assembleur très proche de la machine. Mais à l'époque le logiciel ne bénéfice pas de l'aura qui le caractérise aujourd'hui, il n'était même pas reconnu par les designers de la machine…

La conception de l'AGC débuta dès 1961. Il rejoint la fusée Apollo en 1962. Et se dote de son logiciel, nommé CORONA, et effectue son premier vol en 1966. Les objectifs du programme informatique seront maintenus jusqu'en 1969. Au final, 75 ordinateurs et 138 DSKY seront construits par 2 000 employés de Raytheon venant du programme des missiles Solaris, sur la base du design du MIT. Il sera ainsi à l'époque l'ordinateur construit en plus grand nombre.

L'expérience acquise par la NASA a été considérable, en particulier dans la gestion de projet logiciel en temps réel. Il est difficile de mesurer cet apport, que certains considèrent comme considérable. A l'échelle du coût du programme Apollo lui-même, estimé au début des années 70 à 25 milliards de dollars (soit selon nos calculs environ 150 milliards de dollars aujourd'hui). •

Programme Apollo :
des innovations cruciales pour créer l'informatique moderne

François Tonic

Yves a levé le voile sur la conquête spatiale, le programme Apollo et comment la NASA a réussi à envoyer sur la Lune des « machines » et des hommes. Il a rappelé les énormes défis techniques qu'il a fallu relever, particulièrement sur la partie électronique : inventer un ordinateur transportable dans l'espace et créer les premiers véritables logiciels jamais codés.

Sur la partie fusée et propulsion, nous ne reviendrons pas sur le travail des équipes dirigées par Wernher Van Braun, le père des missiles V2 du IIIe Reich, l'ancêtre des missiles balistiques actuels. Van Braun se livre aux Américains avec plusieurs centaines d'ingénieurs durant les derniers mois de la guerre. Il va alors travailler pour l'armée américaine. Il développe les missiles balistiques du projet Atlas. Le chercheur se penche rapidement sur les lanceurs spatiaux et notamment la fusée Jupiter qui était à l'origine un missile. La mission est simple : mettre sur orbite le premier satellite américain. Chose faite le 1er février 1958. Mais Explorer 1 n'est qu'une réponse à l'URSS et à son Spoutnik, mis en orbite le 4 octobre 1957.

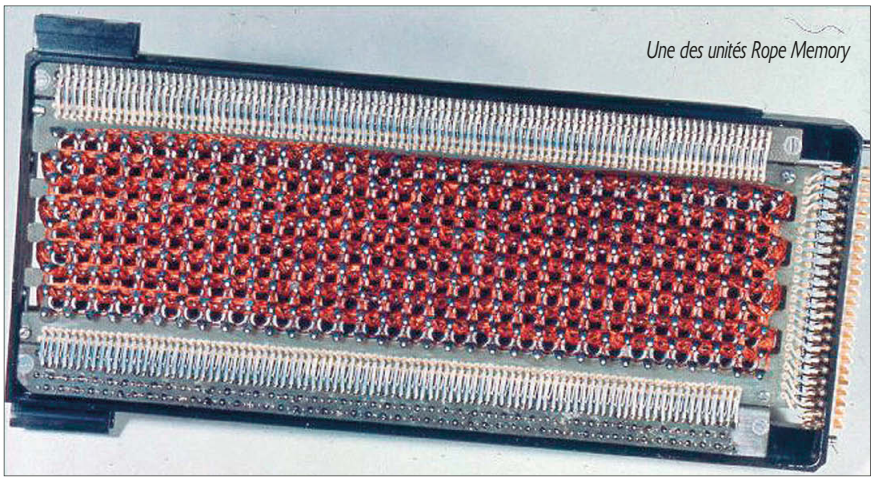
Le projet Saturn, un lanceur lourd, est acté avant le programme Apollo pour permettre à l'armée d'envoyer des satellites plus volumineux. Dès la création de la NASA, en juillet 1958, les responsables considèrent le projet Saturn comme stratégique. Saturn V, 11 ans plus tard, sera l'aboutissement de ce gigantesque projet.

Cette course à l'espace est une compétition URSS – USA. Les soviétiques gagnent une autre manche le 12 avril 1961 avec le premier homme dans l'espace : Gagarine et la mission Vostok 1. Cet exploit sera le tournant des ambitions américaines et du programme lunaire même si les USA imitent l'URSS le 20 février 1962 avec le programme Mercury et John Glenn, premier américain dans l'espace.

Un ordinateur "miniature" et « léger »

Les ordinateurs des années 50 et 60 étaient des monstres par le poids et les dimensions. L'énorme défi était de créer un matériel capable d'être utilisable dans l'espace sans surpoids pour la capsule et le module lunaire. L'ordinateur devait pouvoir exécuter le programme principal et les sous-routines pour réaliser des doubles ou triples opérations. La mémoire fixe (la ROM donc) était vitale car elle embarquait en dur le logiciel, avec, en plus, une mémoire effaçable. La mémoire, appelée rope memory, contenait les programmes, les données, les coordonnées. La mémoire non fixe faisait le lien entre l'interface (saisie et écran) et les programmes / données en mémoire fixe.

On a du mal à imaginer l'incroyable défi technique pour miniaturiser l'ordinateur (en dimensions et en poids) et y installer les programmes. Il a fallu inventer l'ordinateur moderne.



Une des unités Rope Memory

Apollo Guidance Computer (AGC) : le premier ordinateur transportable

Pour définition, un ordinateur possède trois grands composants : le processeur (CPU), la mémoire et les entrées / sorties. Aujourd'hui, quoi que de plus naturel que les données aillent de la mémoire au processeur et inversement, que le processeur traite des millions d'opérations par seconde. Dans les années 60, nous étions très loin des premiers ordinateurs personnels de la fin des années 70.

L'AGC représente donc un exploit technique : 55 cm x 33 cm pour une hauteur de 15 cm, le tout fermé d'une enveloppe impossible à ouvrir. Le tout pesant 32 kg. La puissance de la machine était limitée.

DES JOKES DANS LES COMMENTAIRES

Les développeurs sont parfois facétieux et ceux de la NASA n'ont pas manqué de l'écrire :

- Des références à Shakespeare Henry 6, act 2, scene 4
- Un nom de fichier au nom d'un jeu d'arcade (Pinball)
- Burn baby burn : référence au mouvement Black Power
- Des commentaires qui espèrent beaucoup : temporaire j'espère j'espère j'espère

Le bloc AGC contenait les modules mémoires (ROPE memory), la mémoire effaçable, les capteurs, les modules logiques. A cela se rajoute le DSKY (écran + clavier). En gros, l'ordinateur de bord comportait 7 blocs : timer, générateur de séquences, le "processeur" central, la mémoire, les contrôles prioritaires, les entrées / sorties et l'alimentation.

Les astronautes communiquaient avec l'ordinateur avec le DSKY composé d'un affichage et 19 boutons. L'interaction était basée sur les nombres : 2 nombres pour les verbs (actions / commandes) et nouns (données), 5 pour les données à afficher. Les données étaient affi-

chées sur 3 afficheurs. L'astronaute voyait les verbs, nouns et programmes en cours via les afficheurs PROG, VERB, NOUN. Il pouvait saisir des combinaisons de verbs et de nouns, changer le programme à exécuter. Une centaine de combinaisons étaient utilisables, par exemple : V50N18E.
ou encore :
27 : affiche la mémoire fixe
47 : initialise AGS (via routine 47)
Les programmes avaient un rôle précis durant la mission : le P63 correspondait à la phase de descente vers la surface lunaire, le P64 pour la phase d'approche avec l'atterrissage, etc. Ces programmes étaient exécutés à des positions précises. Les astronautes devaient les lancer via le DSKY.
Les programmes P63 à 67 concernent la phase de descente vers la Lune, l'approche vers la Lune. Les programmes sont regroupés sous l'appellation Pxx, les routines sous le nom Rxx. Les sous-routines sont nommées en Sxx.x
Exploit incroyable, l'ordinateur était capable d'exécuter plusieurs programmes à la fois.

Une mémoire fixée à la main

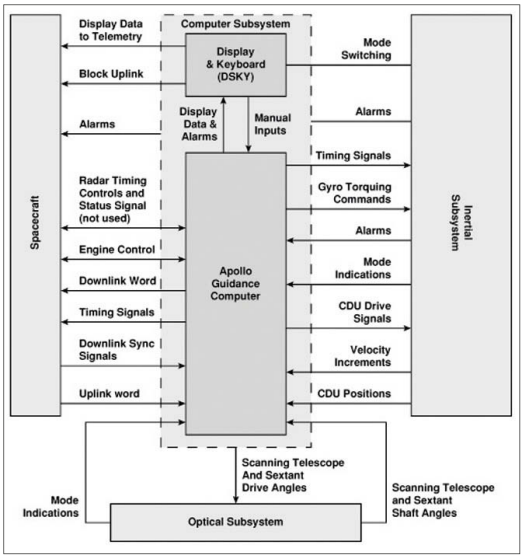
Encore une fois, il faut bien comprendre que les ingénieurs devaient inventer des technologies qui n'existaient pas. La mémoire a été un défi et la NASA sous-estima la taille nécessaire, ce qui compliqua le travail des développeurs.
La Core Rope Memory était la ROM de l'AGC. La technologie fut conçue par le fameux MIT Instrumentation et la société Raytheon fut chargée de la construction des modules. La ROM était construite avec les noyaux en ferrite. Les signaux passaient dans les noyaux par des fils, un noyau pouvait avoir jusqu'à 64 fils. Les noyaux n'étaient pas magnétisés dans un sens ou dans l'autre pour stocker les 0 et 1 (le fameux binaire machine), chaque noyau du rope memory était un transformateur avec des fils passant dans le noyau ou à côté : les fils traversant le noyau stockaient le 1, ceux qui le contournaient étaient le 0.
Pour construire ces modules, des jeunes femmes travaillaient souvent par paire sur une unité mémoire, d'où le surnom de LOL (Little Old Ladies). Elles devaient passer et repasser des mètres de fils de cuivre à travers les noyaux composant le bloc.
L'AGC était aussi un assemblage de circuits imprimés qui étaient encore rudimentaires dans leur design et leur fabrication. Leur coût était très élevé au début des années 60 mais les tarifs baissèrent énormément en quelques années, réduisant la facture de l'AGC. Ces circuits étaient indispensables pour les opérations logiques grâce à l'utilisation de modules logiques. Il s'agissait de petites puces soudées aux circuits. Une technologie nouvelle et peu testée. Le circuit imprimé fut inventé en 1958, le premier circuit disponible sur le marché était le NOR Gate (1961) comprenant 3 transistors. C'était la première fois que les circuits imprimés (et donc des puces électroniques) étaient massivement utilisés.
Bref, une des lignes directrices du projet était d'intégrer dans une puce plusieurs transistors et composants au lieu de les avoir individuellement sur une planche électronique. Les avantages étaient énormes : gain de place, consommation plus basse, coût réduit. C'est le véritable début du processeur au sens CPU et de la miniaturisation. Les ROPE Memory furent soigneusement testées durant plusieurs mois. De nombreux bugs et dysfonctionnements furent corrigés après Apollo 11.

Un bug qui n'en n'était pas un

Durant la phase finale du premier alunissage, l'ordinateur connut un problème de fonctionnement et reboota ! Le pilotage passa alors en mode manuel. Et tout se passa bien même si les astronautes ont dû gérer une situation tendue ; mais l'Apollo Guidance Computer avait parfaitement rempli son rôle.
L'erreur, et l'alarme qui raisonna (alarmes 1201 et 1202), ne venait pas des programmes. L'ordinateur reçut des données du radar de rendez-vous (utilisé pour le retour vers le Command Module) qui fut activé lors de l'atterrissage, en cas d'annulation à la dernière minute de l'alunissage. Cette action envoya de nouvelles données à l'ordinateur ce qui provoqua un dépassement de capacité. L'ordinateur était conçu pour traiter les tâches prioritaires et supprimer toutes les autres.
Finalement, l'ordinateur fonctionnait comme il avait été programmé et conçu. Et il n'a jamais connu de bugs durant la phase de descente. L'erreur venait de la procédure employée durant la descente et l'activation du radar de rendez-vous.

Un code disponible sur Github

Le code source des programmes de la mission atteint presque 2 mètres de hauteur ! Ce code fut une première fois scanné en 2003 mais plusieurs pages furent endommagées et des lignes de codes illisibles. Il a fallu corriger et recréer les codes perdus. La publication publique sur Github a été l'oeuvre d'un employé de la NASA, Chris Garry. On trouve le code pour les ordinateurs du Command Module et du Lunar Module. Plusieurs versions d'assembleur furent utilisées. Site : https://github.com/chrislgarry/Apollo-11/
L'ordinateur et les programmes furent progressivement testés dans les missions Apollo successives. L'ultime vérification eut lieu avec Apollo 8 qui fit le tour de la Lune (fin 1968). Les derniers programmes pour l'alunissage furent codés et installés quelques mois après mais sans que tout ait été testé.
Souvenez-vous qu'à l'époque, les programmeurs n'avaient pas les outils de tests que l'on a maintenant. Il fallait relire et relire les codes, les corriger et refaire l'opération. Les programmes étaient ensuite testés en conditions réelles dans un simulateur. La pression était sans aucun doute incroyable : car si le code ne fonctionnait pas comme prévu, les astronautes mourraient.



Architecture de l'ordinateur de bord

Pour aller plus loin :

F. O'Brien, ***the apollo guidance computer,*** Springer, 2010
Pour construire un AGC :
http://www.ibiblio.org/apollo/Pultorak_files/
(voir notamment les fichiers build_agc) •

FICHE TECHNIQUE

- 36 000 mots sur la ROM
- 2 000 mots sur la RAM
- 16-bit pour la longueur des mots
- Vitesse d'horloge (oscillateur) : 2 048 Mhz

La vitesse d'horloge était divisée par 2 pour la fréquence de calcul (1 024 Mhz) et encore par 2 (soit 512 Mhz) pour assurer la synchronisation des systèmes d'Apollo.

Le développeur full-stack est mort !

Vive le développeur universel !

A la fin du 18ème siècle, le secteur agricole représentait encore l'activité principale de 85% de la population. Et oui, il y a tout juste deux cents ans, tout le monde était agriculteur full-stack et le visage économique du monde et de la France était totalement différent.

Par Thomas Gx, fondateur de CommitStrip et d'Oh My Bot !

C'était il y a huit générations. Cela signifie que les arrière-grands-parents de nos arrière-grands-parents, des personnes intuitivement pas si éloignées de nous dans le temps, ont vécu une vie à des années lumières de ce que nous vivons. Leurs préoccupations du quotidien étaient tournées vers la récolte, la météo, les maladies, les guerres.

Pour ma part, comme sans doute pour une bonne partie d'entre vous, tous mes ancêtres étaient des paysans, jusqu'à mes grands-parents, qui étaient de simples agriculteurs, tout juste motorisés et pas du tout automatisés. C'est seulement mon père qui a rompu la tradition de centaines d'années de paysannerie. J'imagine déjà certains lecteurs me vanner : « Moi, dans ma famille, ça fait deux cents ans qu'on est informaticien de père en fils, mon arrière arrière grand-père codait en PHP2 ».

Si je vous parle d'histoire et d'agriculture aujourd'hui, ce n'est pas pour vous remémorer vos cours d'histoire du collège, mais bien pour réfléchir à nos métiers. En réfléchissant à notre condition de développeur, full-stack ou non, à notre rôle dans l'industrie, dans la société et dans l'histoire, je me suis demandé à quel point, pour un historien du 25ème siècle, les codeurs de notre temps auront joué un rôle clé.

Ce n'est pas forcément notre réflexe à nous autres que de puiser dans l'histoire pour appréhender notre nouvelle révolution industrielle, nous qui nous réclamons volontiers modernistes, futuristes, progressistes, avides de prospectives.

Révolution industrielle ou pas d'ailleurs ? De quoi parle-t-on quand on parle de révolution industrielle ? Comment des découvertes scientifiques peuvent-elles à ce point bouleverser la société qu'on parle alors de « révolution » ?

Première révolution industrielle

C'est après la révolution, début du 19ème siècle, que commence vraiment notre ère industrielle. A cette époque, la France souffre en fait d'un handicap qui l'empêche de connaître la même croissance que le Royaume Uni : notre essor démographique reste modéré, et donc la demande reste moindre. Incapable d'être vraiment compétitive (eh oui, le problème se posait déjà !), la France se concentre sur les exportations pour lesquelles l'importance de la qualité lui donne un avantage hors-prix : en effet, la tradition industrielle française à doter le pays d'une main-d'œuvre qualifiée lui permet de faire la différence avec ses voisins.

Alors qu'actuellement, la Silicon Valley s'arrache les talents français, c'est remarquable de voir que la réputation de « Qualité » de la France ne date pas d'hier.

Eh oui, encore aujourd'hui, les Français ont la réputation de rechercher et d'exiger la qualité et l'excellence. Lors de mes expatriations en Asie pour développer mon agence web, je me souviens avoir échangé avec un négociant de marbre indien, dans le sud de l'Inde, qui s'était écrié, après que je me sois présenté en tant que français : « Ah les Français. Ils veulent de la qualité avant tout. Ce n'est pas comme les chinois, qui veulent d'abord des prix bas. Vous les Français, vous voulez de la qualité, peu importe le prix. » Et ça sonnait plutôt juste à mes oreilles, moi qui

essayais alors, avec une certaine difficulté, d'obtenir du code de bonne qualité de mes premiers développeurs indiens. Mais c'est une autre histoire, que je vous conterai peut-être un jour.

Revenons à notre Première révolution industrielle, qu'on considère voir décoller en France entre 1820 et 1840 : la machine à vapeur se développe de plus en plus au sein des industries, et remplace peu à peu l'énergie hydraulique. Il faut du fer pour fabriquer les machines, du charbon pour alimenter la machine à vapeur et s'ensuit donc une course à l'approvisionnement en produits miniers et une recherche effrénée de procédés toujours plus optimisés. Dans la foulée, la vapeur, le fer et le charbon vont permettre le développement des chemins de fer, qui vont eux-mêmes relancer l'industrie. Les transports ferroviaires ou maritimes vont en effet donner naissance à de nouveaux marchés.

On discerne bien les grandes conséquences sociétales qui naissent de cette révolution industrielle. Dans ce contexte, les centres urbains vont se créer et se développer autour des usines et des ateliers. C'est le début de l'exode rural : l'installation des différentes entreprises en un même lieu permet d'améliorer la rentabilité des productions. Les progrès techniques sont toujours renouvelés, augmentant sans cesse le rendement industriel. Voyez comme quelques inventions et une nouvelle énergie peuvent complètement changer le modèle de la société en une ou deux générations.

Deuxième révolution Industrielle

Alors que la production mondiale avait mis 120 ans pour doubler entre 1700 et 1820, l'apparition et le développement de nouvelles techniques permettent un premier doublement en cinquante ans entre 1820 et 1870, puis un second doublement, en quarante ans, entre 1870 et 1910.

Mais déjà un nouveau grand bouleversement allait arriver : l'électricité. Alors que la première révolution industrielle repose sur le charbon, la métallurgie, le textile et la machine à vapeur, la seconde trouve ses fondements dans l'électricité, la mécanique, le pétrole et la chimie. La découverte de mines d'or en Europe et aux Etats-Unis permet également de relancer la production industrielle des pays concernés.

Le moteur électrique se généralise. Dans les ateliers et les usines de la fin du 19ème siècle et du début 20ème, il est encore encombrant et lourd mais il va rapidement remplacer le moteur à vapeur en permettant un meilleur partage de la force motrice au sein des ateliers.

Outre les économies de matériel et le gain en sécurité, l'électricité donne une liberté nouvelle pour rationaliser l'organisation spatiale des usines de façon strictement conforme à la succession des étapes de la fabrication. Autrement dit, la taylorisation du travail et les chaînes de montage – apparues en Amérique dès les années 1910 – sont en grande partie filles de l'électricité.

Encore une fois, des inventions bouleversent notre société. L'ère de l'artisanat et des micro-entrepreneurs fait place à la mécanisation du travail, à la naissance d'une classe ouvrière et fonde le capitalisme.

L'économie des pays d'Europe et d'Amérique du Nord sera alors considérablement bouleversée par le développement de l'industrie lourde et des nouveaux secteurs industriels naissants tels que celui de l'automobile.

Au 20ème siècle, grâce surtout à l'utilisation de combustibles fossiles, les activités industrielles ont été multipliées par cinquante, alors que la population mondiale a triplé, et que le volume de l'économie mondiale a été multiplié par vingt, et la consommation de combustibles fossiles par trente.

La seconde révolution industrielle prend fin à l'aube de la seconde guerre mondiale.

Troisième révolution industrielle

En deux cents ans et deux révolutions industrielles, le visage de notre société a totalement changé : le pourcentage de la population vivant de l'agriculture est passé de 85% à 5% à la fin du 20ème siècle !

C'est justement en cette fin de 20ème siècle que naît notre fameuse révolution informatique, d'ores et déjà considérée comme la troisième révolution industrielle. Mais est-ce que cette révolution à laquelle nous participons aura autant d'impact sociétal que les deux premières ?

Cette troisième révolution industrielle est celle de l'électronique, des télécommunications, de l'informatique, de l'audiovisuel et du nucléaire. Ils rendent possible la production de matériels miniaturisés, de robots et de l'automatisation poussée de la production, le développement des technologies spatiales et celui des biotechnologies.

A mon sens, l'avènement de l'informatique et d'Internet est encore plus bouleversant que les autres révolutions. Et c'est nous, codeurs, développeurs, inventeurs de notre temps, qui en sommes les grands acteurs !

Mais c'est quoi être codeur, développeur ? Et plus particulièrement, c'est quoi être développeur full-stack ? Ce fameux mouton à cinq pattes que tout le monde cherche mais que personne ne trouve.

Le développeur full-stack se réveille, il lit ses emails, il trie ses newsletters, qu'il lira consciencieusement pendant les transports en commun. Puis il codera pendant la journée, en prenant bien sûr soin de consulter pendant sa pause déjeuner les blogs qu'il a mis en favoris.

Le soir venu, ça sera tuto, coding, Quora, Stackoverflow et commit sur les side projects sur Github.

Tout cela pour tenter de rester à jour. De garder le rythme. D'être expert. Mais expert en quoi ?

En développement back-end ? Bien sûr, c'est la base, même si ce n'est pas évident de savoir sur quelle techno s'engager.

Le Front-end ? Evidemment, le JS, c'est incontournable ces jours-ci et c'est plutôt simple. Encore que, migrer d'Angular à React, ce n'est pas trivial ni plaisant.

Mais quid du dev natif mobile et du responsive? Sur iOS et Android bien sûr.

Quid aussi de l'Internet des Objets ? Et la blockchain ?

Et tout ce qui est Devops ? L'intégration continue ? La conteneurisation ? L'orchestration ? Les grids ? Les protocoles de sécurité ?

Ah ! Et j'oubliais le machine learning, le deep learning, la gestion du langage naturel,

l'IA. Bref, réglons définitivement la question du développeur full-stack : il n'existe plus. Le full-stack fait référence à une époque révolue où nous envoyions du PHP/HTML basique en FTP sur une stack LAMP. Mais c'est fini.

A vrai dire, c'est assez courant de définir un développeur par les technologies qu'il utilise. C'est même ainsi que nous nous présentons souvent. Ce qui peut faire sens pour expliquer concrètement ce que nous faisons, mais ce qui réduit la portée du codeur à sa simple capacité à écrire du code, à transcrire des spécifications en application.

Aujourd'hui, un développeur ne devrait pas tenter de devenir full-stack, il doit devenir un développeur universel !

Le vrai développeur complet de notre époque ne doit pas se contenter de coder, mais il doit également penser plus largement à sa création de valeur.

Ce statut va impliquer une sorte de grand écart. D'un côté, le développeur doit d'abord être spécialisé pour être compétitif. Même dans un environnement de pénurie en ressources et donc favorable aux développeurs, nous restons dans une économie mondialisée où n'importe quelle personne avec un ordinateur et une connexion peut prendre votre place, à des coûts souvent moindres. D'où la nécessité d'être excellent et de faire la différence dans un domaine.

De l'autre côté, le développeur doit également faire la différence en ayant une vraie compréhension du marché et de la création de valeur.

Ce n'est pas un hasard si les cinq plus grosses capitalisations boursières du monde (Apple, Google, Microsoft, Amazon, Facebook) sont des boîtes technologiques et même, principalement software. Certes, il y a le cas particulier d'Apple qui tire encore profit du hardware, mais la réalité est que la valeur se crée avec le software aujourd'hui. Bien peu d'entre nous feront fortune avec du hardware.

Quelle est la différence entre un objet basique et un smart object ? La différence, c'est du temps de développeur. Rendre smart un objet, c'est ni plus ni moins que demander à un codeur de passer du temps dessus.

Je citais les deux premières révolutions industrielles comme exemple de la façon dont ces bonds provoquent des changements et des remises en question de l'organisation de la société.

Si les politiques publiques peuvent encourager des pans de l'économie (comme la naissance des brevets qui a favorisé la première révolution industrielle), c'est bien nous qui devons penser et conceptualiser le pacte social de demain.

A ce titre, inspirons-nous des hommes d'esprit universels comme Léonard de Vinci, Descartes, Newton, Marx, Einstein, qui ne réduisaient pas l'intelligence à la seule connaissance des sciences ou de la technique, mais qui dissertaient grandement sur la société, la philosophie, et l'art ! Trop peu d'ingénieurs et de développeurs prennent la parole sur les bouleversements qu'ils provoquent dans la société.

Il est important de le faire, car si le présent pose des questions, ce qui arrive en soulève encore davantage pour tout le monde, y compris pour nous.

Quatrième révolution industrielle

Après la mécanisation, l'industrialisation, l'automatisation, l'idée d'une quatrième révolution industrielle mêle collaboration des systèmes et intelligence artificielle.

C'est celle qui va voir aboutir une Intelligence Artificielle Forte. Elle va arriver, c'est inéluctable au vu des efforts entrepris par les meilleurs ingénieurs partout à travers le monde.

Paradoxalement, ce sont les métiers intellectuels qui vont d'abord être touchés et qui vont disparaître les premiers. Les comptables, médecins et autres avocats, tous les métiers qui consistent à traiter en entrée de la donnée, à l'analyser et à renvoyer un diagnostic ou une étude vont voir leur création de valeur très diminuée par l'IA. Comment un radiologue ou un avocat peut-il espérer être plus efficace qu'un programme qui va prendre en compte l'immensité du big data ? C'est déjà délirant de s'en remettre à un seul avocat qui va seulement invoquer son cerveau pour analyser un dossier à l'aune des quelques bribes de mémoire qu'il réussit à rassembler.

A court terme, c'est-à-dire d'ici dix à vingt ans, la plupart des métiers intellectuels vont disparaître et muter. Comment ? Par qui ?

Ben, par nous. Les développeurs et les ingénieurs informatiques. Et d'ailleurs, nous parlions des compétences et des expertises du développeur : eh bien il y a fort à parier qu'une bonne partie d'entre nous finira par se concentrer sur ce qui est complémentaire à l'IA. Car les interfaces et les systèmes sur lesquels nous travaillons aujourd'hui (écran, navigateur, applicatif) vont bien changer.

Qui va les remplacer ? Ce sont les assistants. Les assistants virtuels vont tout balayer. Combien de temps pensez-vous que vont survivre les sites de recherche ? Pendant combien de temps allons-nous encore développer des formulaires sur des sites web chargés sur des smartphones à l'écran minuscule ? Google a d'ores et déjà annoncé le début de l'ère de l'Assistance.

Beaucoup de nos interactions avec la machine passeront par la voix. C'est d'ailleurs ce que prédit très justement Satya Nadella, CEO de Microsoft, quand il dit « Voice is the next platform ». A ce sujet, le film Her en est une excellente illustration.

Alors, c'en sera bientôt fini des interfaces de recherche type la Fourchette, Booking ou Liligo. La réelle valeur ajoutée sera le moteur de recommandation basé sur la connaissance parfaite de l'utilisateur par l'assistant virtuel.

Les usages vont donc évoluer inéluctablement et tout laisse à penser qu'un certain nombre d'interfaces deviendront obsolètes. Une grande demande va voir le jour dans les années à venir sur les programmes capables de se connecter aux grandes plateformes d'assistant.

Qui sont-ils, ces assistants ? Nous les connaissons déjà

bien : ils s'appellent Amazon Alexa, Google Assistant, Apple Siri ou Microsoft Cortana, Facebook M. Très bientôt, l'enjeu principal du développeur chez Booking, par exemple, va être d'interconnecter ces assistants avec son propre service.

Cela pose d'ailleurs un bien grand nombre de questions cruciales : quid de la data ? A qui appartient-elle ? Est-ce que des assistants virtuels open source, indépendants des GAFAMs et des BATX mais autant performants, arriveront à percer ?

Et comment gérer la publicité ? Est-ce que nos assistants se feront berner par des placements de produits ? Qui décidera pour nous ? Déjà aujourd'hui, l'ordre d'arrivée des résultats de recherche sur Google, Booking, etc, est bien souvent biaisé. Mais plus nous déléguons, plus cette question est importante et plus notre exigence en termes de transparence sera grande.

Ce sont des questions éthiques, sociales, philosophiques sur lesquelles chacun d'entre nous, développeurs, devrait avoir une opinion et être expert, car nous serons amenés à y participer et même à en être victime. Eh oui nous sommes tout autant concernés que les autres professions, il suffit de réfléchir au sort des développeurs front-end. On évoque les avocats et les comptables, mais on peut prédire que la plupart des développeurs front-end de 2017 devra se réinventer avant longtemps.

Le rôle du développeur dans l'ASI ou l'IA Forte

Nous parlions jusqu'ici du relatif court terme. Mais si on passe à trente ou quarante ans, c'est le moment d'évoquer le futur de l'IA.

L'apogée de la quatrième révolution industrielle pourrait venir de l'association de la diffusion massive d'ordinateurs de performance supérieure à l'avènement d'une Super Intelligence Artificielle. Une bonne partie d'entre nous pourrait donc en être contemporain. Quelles seront les conséquences sur la société de cette quatrième révolution et l'avènement de l'IA globale et super intelligente ?

Les besoins de base des humains seront potentiellement tous gratuits : l'énergie automatisée sera gratuite et illimitée, la nourriture sera disponible en abondance grâce aux champs de cultures automatisés, et les maisons seront construites automatiquement pour presque rien par des imprimantes 3D géantes.

Les maladies seront pratiquement toutes vaincues, voire le vieillissement vaincu.

Un changement si radical est-il possible sur une si courte période ? Ce n'est pas du tout exclus si on considère que le contexte n'a jamais été aussi propice : le monde connaît actuellement un niveau de richesse et de connaissance qui est bien plus élevé que lors des deux premières révolutions industrielles, et parallèlement un niveau de guerre, de pauvreté ou de maladie qui n'a jamais été aussi faible dans l'histoire.

Comme lors les révolutions industrielles précédentes, notre modèle de société sera sinon bouleversé, au moins profondément transformé par ces automatisations. La société ne sera plus dirigée vers le travail. Mais alors par quoi ? Que ferons-nous de nos journées ?

Il se pourrait bien qu'une grande majorité de la population, peut-être 95%, devienne les nouveaux prolétaires, se contentant du pain et des jeux, vivant une vie probablement paisible mais extrêmement contrôlée, pour maintenir un équilibre de paix et de prospérité pour tous. Le revenu universel, si discuté lors de la dernière campagne présidentielle française, deviendrait alors inutile. Nous assisterions alors à un exode urbain, puisque s'entasser dans des mégalopoles étriquées et polluées serait alors devenu sans intérêt. Nous retrouverions une grande décentralisation et des communautés plus petites.

Avec peut-être, paroxysme du capitalisme, quelques personnes ultra-riches qui contrôleraient les IA et tiendraient le monde tout en jouissant d'un confort et d'une puissance particulièrement inouïs. Une situation qui pourrait malgré tout être la moins pire pour la paix de la planète.

Il nous serait avisé, à nous, développeurs, de penser à notre rôle dans l'avènement de ce futur.

Car c'est nous qui allons, dans les années à venir, poser les bases des réflexions autour de ces évolutions. C'est nous qui allons créer les entreprises d'IA de demain.

Tout d'abord, se former : parce que nous allons probablement tous finir par travailler sur du deep learning et du machine learning, c'est quasi sûr, et on serait bien inspirés de s'y mettre rapidement.

Aparté : Si vous avez envie d'apprendre le machine learning, il faut bien commencer par quelque part. Voici donc trois livres que vous pouvez vous mettre à lire dès ce soir :

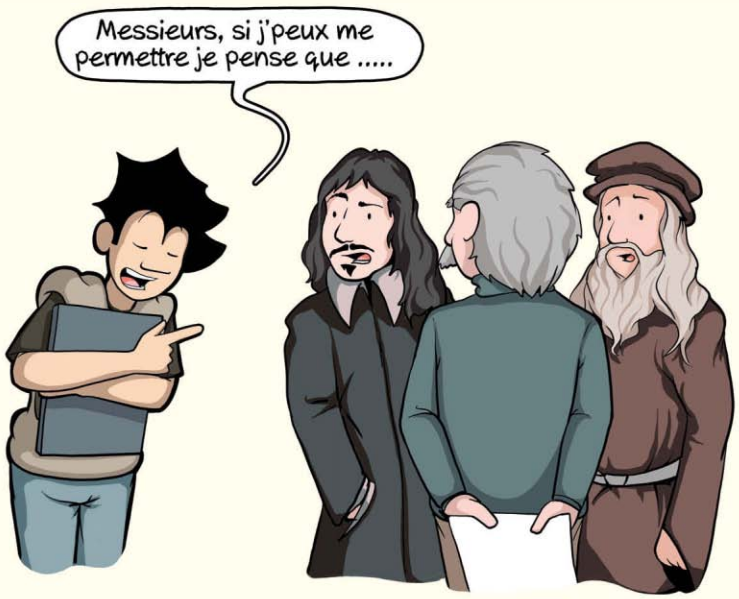
- Hands-On Machine Learning with Scikit-Learn and TensorFlow par Aurélien Géron
- Make Your Own Neural Network par Tariq Rashid
- Artificial Intelligence : A Modern Approach par Russel & Norvig

(En anglais uniquement, que tout bon développeur universel pratique ;)).

Ensuite, le développeur universel devrait être l'explorateur et le philosophe de notre temps. Un développeur aujourd'hui devrait associer à son savoir une réflexion philosophique, sociale, économique, juridique à ses travaux.

Car on ne peut pas se permettre de laisser l'organisation de la société à des personnes qui ne comprennent pas la technologie.

Une nouvelle génération de développeurs doit sortir de son carcan technique et s'exprimer sur le nouveau socle social qui va devenir le nôtre. Il y a trop peu de développeurs et d'ingénieurs informaticiens en politique, alors que l'IA est un enjeu géopolitique critique dès aujourd'hui.



Alors si je résume le développeur universel en 2017, c'est un développeur qui doit être expert en IA, philosophe, économiste, politicien et écrivain.

Bon… Eh bien on va encore moins dormir ce soir !

Pascal UCSD sur **Apple II**

Partie 2

• Alain Zanchetta
Software Development Engineer dans l'équipe **Microsoft HoloLens.**

© Mystère Martin

Alors qu'il a été le premier ordinateur personnel à proposer des graphismes en couleur et haute résolution, l'Apple II a rapidement été surclassé dans ce domaine en particulier, et dans tout ce qui touche au jeu en général par les machines plus récentes comme le Commodore 64. En revanche, il y a un domaine où l'Apple II n'a été égalé que bien plus tard, et ce, uniquement par des ordinateurs bien plus puissants, c'est dans la variété des langages de programmation disponibles. Nous allons nous intéresser aujourd'hui au Pascal UCSD, créé par l'Université de Californie de San Diego et diffusé par Apple. ***Suite et fin de l'article.***

Utilisation de la mémoire

La taille d'une procédure ou d'une fonction est limitée à 1200 octets de code machine et les variables locales sont limitées à 30 Ko environ… Ce qui est très large compte tenu de la taille mémoire de l'Apple II lui-même (on verra plus loin comment découper un programme en différentes parties permet d'exécuter un volume de code bien plus important que ce la mémoire de l'Apple préfigure).

Dans l'exemple ci-dessus :

- Calc occupe 54 octets de code et 4 octets de données ;
- FactTest occupe 95 octets de code et 20 octets de données ;
- Le « main » occupe 16 octets de code uniquement.

A l'exécution, la fonction MEMAVAIL retourne le nombre de mots (2 octets) disponibles.

```
program MemorySample;

type list=^node;
    node = record
       alloc : integer;
       memory : integer;
       next : list;
    end;

var i : integer;
   head, tail, temp : list;

begin
   head := nil;
   tail := nil;
   for i:=1 to 5 do
   begin
      new(temp);
      temp^.alloc := i;
      temp^.memory := memavail;
      temp^.next := nil;
      if head = nil then head := temp;
      if tail <> nil then
         tail^.next := temp;
      tail := temp;
   end;
   temp := head;
   while temp <> nil do
   begin
      write('After ');
      write(temp^.alloc);
      write(' allocations, available memory = ');
      writeln(temp^.memory);
      temp := temp^.next;
   end;
end.
```

L'exécution de ce programme qui illustre en passant la notion de pointeur Pascal montre l'évolution de la mémoire : chaque allocation de nœud représente trois mots, soit six octets : deux octets pour chaque entier et deux octets pour le pointeur.

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ? [1.2]
Running...
After 1 allocations, available memory = 19197
After 2 allocations, available memory = 19194
After 3 allocations, available memory = 19191
After 4 allocations, available memory = 19188
After 5 allocations, available memory = 19185
```

Puisqu'on parle de mémoire : à partir de la version 1.2, le Pascal UCSD supporte 128 Ko de mémoire : pour y accéder, il faut créer une disquette de démarrage en remplaçant le fichier SYSTEM.APPLE de « Apple1: » par le fichier 128K.APPLE présent sur la disquette « Apple3: » et le fichier SYSTEM.PASCAL par le fichier 128K.PASCAL : l'écran de démarrage confirme la modification :

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ? [1.2]


Welcome APPL128, to Apple II Pascal 1.2
```

```
Based on UCSD Pascal II.1

Current date is 17-Mar-81

Pascal system size is 128K

Copyright Apple Computer 1979,1980,1983
Copyright U.C. Regents 1979
```

A l'exécution, on voit qu'un peu plus de mémoire est disponible :

```
Command: E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, ? [1.2]
Running...
After 1 allocations, available memory = 22268
After 2 allocations, available memory = 22265
After 3 allocations, available memory = 22262
After 4 allocations, available memory = 22259
After 5 allocations, available memory = 22256
```

La différence est d'environ 3000 mots soit 6 Ko seulement : en fait, c'est surtout la mémoire réservée au programme lui-même qui va être étendue grâce aux 64Ko supplémentaires comme le montrent les schémas suivants : [3].

En réalité, même sur un Apple II limité à 64 Ko de mémoire, il est possible d'écrire des programmes de taille assez importante en s'appuyant sur un mécanisme de chargement / déchargement du code permettant de n'avoir en mémoire que le code « actif ». Cette gestion s'appuie sur la notion de segment : pour appeler un sous-programme, le segment de code qui le contient doit être présent en mémoire, mais une fois le sous-programme exécuté, ce même segment n'est généralement plus nécessaire et peut être remplacé par un autre segment. Cela nuit à la rapidité d'exécution mais ce compromis est parfois nécessaire, et Pascal apporte ici des solutions plus élégantes et plus puissantes qu'en Basic.

Par défaut, tout le code d'un programme constitue un seul segment mais en insérant le mot clé SEGMENT devant FUNCTION ou PROCEDURE, on en crée un nouveau, qui contient uniquement cette fonction ou procédure. Cette approche est particulièrement adaptée aux programmes où les différents sous-programmes sont chargés dans des phases bien précises de l'exécution, comme une saisie de données importante au démarrage suivie par une phase de calculs et enfin une phase de présentation des résultats.

Une autre approche, un peu plus compliquée à mettre en œuvre, consiste à créer des sortes de bibliothèques de code appelées unités – mot clé UNIT – qui seront, elles aussi, chargées et déchargées selon les besoins du programme. Le source de chaque unité est structuré en deux parties, l'interface et l'implémentation : les déclarations présentes dans l'interface définissent ce qui est accessible à l'extérieur de l'unité, la partie implémentation peut contenir des variables globales ou des sous-programmes privés ainsi qu'un bloc d'initialisation.

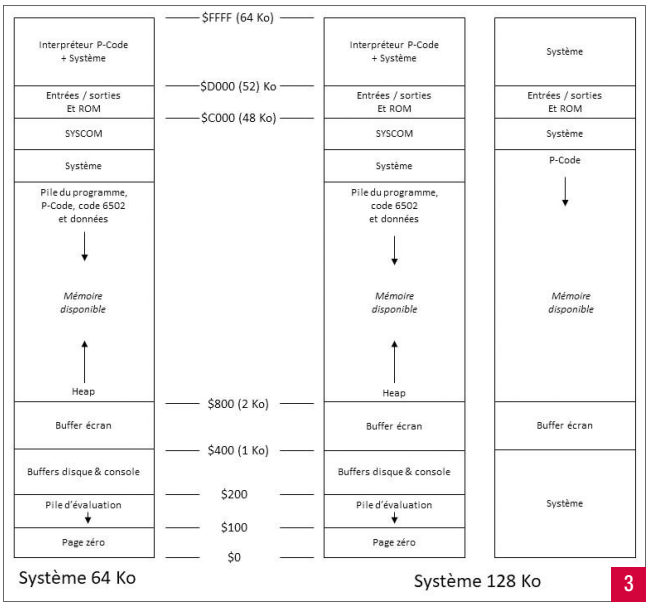
Découpons par exemple le programme de graphismes présenté précédemment en trois unités séparées – une pour chaque sujet – et un programme principal appelant ces trois unités.

Voici l'implémentation de l'unité GraphStar :

```
unit GraphStar;

interface
  uses Transcend, TurtleGraphics;
  procedure DrawStar(radius : integer; centerX : integer; centerY : integer);

implementation

procedure DrawStar;
var i, n, x, y : integer;
  angle : real;
begin
  n := 0;
  PenColor(none);
  for i:=0 to 24 do
  begin
    angle := n * 6.2831 / 24;
    x := centerX + round(radius * cos(angle));
    y := centerY + round(radius * sin(angle));
    MoveTo(x,y);
    PenColor(Orange);
    n := n + 13;
  end;
end;

begin
end.
```

Après avoir structuré de manière similaire GraphText et GraphShape, on peut réécrire le programme GraphDemo de la manière suivante :

```
program GraphMain;

(*$S+*)
(*$N+*)

uses Transcend, TurtleGraphics,
 (*$U #4:GraphStar.code*) GraphStar,
 (*$U #4:GraphText.code*) GraphText,
 (*$U #4:GraphShape.code*) GraphShape;

var s : string;
```



3

```
begin
  InitTurtle;
  DrawStar(50, 51, 102);
  DrawMessage('Hello world', 100, 150);
  DrawShapes(100, 120, 8);;
  read(s);
end.
```

Pour compiler ce fichier source, il faut avoir préalablement compilé les différentes unités qu'il utilise. Ensuite, il faut lancer l'éditeur de liens (L(ink dans le menu principal) et saisir un à un les fichiers de code à regrouper. A chaque unité correspondra un segment de code et donc toutes les unités ne seront pas nécessairement chargées en mémoire en même temps (c'est le rôle de la directive (*$N+*) en tête du programme. Ces unités sont à rapprocher des bibliothèques statiques sous Windows car leur code est incorporé dans chaque programme qui les utilise. L'équivalent de la notion de DLL existe mais est très limité : il est possible de compiler une unité en mode partageable à l'aide du mot clé INTRINSIC. Elle peut alors être insérée dans la bibliothèque SYSTEM.LIBRARY et être utilisée par différents programmes sans devoir être incorporée aux fichiers .CODE de ceux-ci. Cela permet en outre de modifier l'implémentation de l'unité partagée sans avoir à recompiler tous les programmes qui l'utilisent.

Quid des performances ?

Disons-le tout de suite, le Pascal UCSD n'a pas de turbo et ses performances ne sont pas extraordinairement meilleures que celles du Basic. Voici deux programmes de tests permettant de comparer les performances des deux langages.

Le premier programme calcule des approximations de factorielles de nombres généralement inaccessibles aux calculs naïfs : il s'appuie sur la relation Log(ab)=Log(a) + Log(b) pour calculer Log(n!)=Log(n) + Log(n-1) + … + Log(2). En utilisant le logarithme en base 10, la partie entière du nombre obtenu est le nombre de chiffres ; quant à la partie décimale, elle correspond au logarithme de la mantisse. Evidemment, il y a des erreurs d'arrondi mais le résultat final n'est pas si mauvais :

```
program Perfs;

  uses applestuff, transcend;

procedure RealFact;
  var n, i, e : integer;
    fact, fract : real;
begin
  Note(100, 30);
  n := 300;
  fact := 0;
  for i:=2 to n do
    fact := fact + Log(i);
  write(n);
  write('! = ');
  e := trunc(fact);
  fract := fact - e;
  write(exp(fract * ln(10)));
  write('E');
  writeln(e);
  Note(100, 30);
end;

begin
  RealFact;
end.
```

```
Running...
300! =  3.06040E614
```

(Bing donne 3.06057512E614)

Voici le programme Basic sensiblement équivalent :

```
10  PRINT  CHR$ (7)
20 N = 300
30 F = 0
40 L10 =  LOG (10)
50  FOR I = 2 TO N:F = F +  LOG (I): NEXT
60 F = F / L10
70 E =  INT (F)
80 FR = F – E
90  PRINT N;"! = "; EXP (FR * L10);"E";E
100  PRINT  CHR$ (7)
```

```
]RUN
300! = 3.06058243E614
```

Le tableau ci-dessous compare les temps d'exécution de ces deux programmes (sur un Apple IIe à 1 MHz) :

	Pascal	Basic Applesoft
Temps d'exécution	12.5 s	7.5 s

Essayons un programme un peu plus riche : il charge dans un tableau de chaînes de caractères un fichier contenant les 47 films réalisés par Woody Allen triés par ordre chronologique, et effectue un tri de ces titres par ordre alphabétique. Le tri utilisé est un Selection Sort qui présente l'avantage d'être aisément transposable en basic.

```
program Selection;

uses applestuff;

var woody : text;
  i,j,n,best : integer;
  lines : array[1..100] of ^string;
  temp : ^string;

begin
  note(100, 50);
  { Turn off error checking so that program will do it }
  (*$I-*)
  reset(woody, 'woody.text');
  if ioresult <> 0 then begin
    writeln('Unable to open file');
    exit(program);
  end;
  { Turn on error checking }
```

```
(*$I+*)
n := 1;
while not(eof(woody)) do
begin
  new(lines[n]);
  readln(woody, lines[n]^);
  n := n + 1;
end;
n := n - 1;
write(n);
writeln(' lines');
note(100, 50);
for i :=1 to n-1 do
begin
  best := i;
  for j:=best+1 to n do
    if lines[j]^ < lines[best]^ then
      best := j;
  if best <> i then
  begin
    temp := lines[i];
    lines[i] := lines[best];
    lines[best] := temp;
  end;
end;
note(100, 50);
for i :=1 to n do
  writeln(lines[i]^);
note(100, 50);
end.
```

Le programme Basic équivalent est le suivant :

```
10  PRINT  CHR$ (7)
20 D$ =  CHR$ (4):F$ = "WOODY.TXT"
30  PRINT D$"OPEN "F$
40  PRINT D$"READ "F$
50  DIM T$(100):N = 0
60  ONERR  GOTO 90
70  INPUT T$(N):N = N + 1
80  GOTO 70
90  PRINT D$"CLOSE "F$: ONERR  GOTO 0
100  PRINT N" TITLES"
110  PRINT  CHR$ (7)
120 N = N - 1
130  FOR I = 0 TO N
140 BE = I
150  FOR J = I + 1 TO N
160  IF T$(J) < T$(BE) THEN BE = J
170  NEXT
180  IF BE < > I THEN A$ = T$(BE):T$(BE) = T$(I):T$(I) = A$
190  NEXT
200  PRINT  CHR$ (7)
210  FOR I = 0 TO N: PRINT T$(I): NEXT
220  PRINT  CHR$ (7)
```

Le tableau ci-dessous compare les temps d'exécution des différentes parties de ces deux programmes :

	Pascal	Basic Applesoft
Chargement du fichier	19.2 s	2.5 s
Selection Sort	3.7 s	9.5 s
Affichage	3.3 s	2.6 s

Bien que DOS 3.3 n'ait jamais eu la réputation d'être rapide (de nombreux DOS alternatifs proposaient des performances bien meilleures tout en préservant la compatibilité avec DOS 3.3), le chargement du fichier est infiniment plus rapide qu'en Pascal. En revanche, le tri lui-même est nettement plus rapide en Pascal.

CONCLUSION

Bien qu'il soit un peu déroutant de prime abord – surtout à une époque où le Basic était disponible immédiatement après l'allumage de l'ordinateur – le système UCSD est au final un des environnements de programmation les plus complets de l'Apple II, avec en particulier un vrai éditeur plein écran et un éditeur de liens facilitant l'écriture de programmes structurés complexes. Outre Pascal, plusieurs autres langages furent disponibles dans cet environnement – Fortran, Prolog ainsi que Modula-2 (inventé comme Pascal par Niklaus Wirth) - et le partage de l'éditeur, du gestionnaire de fichiers et d'une manière générale de cette navigation par menus en facilitait la prise en main.

Si le Pascal UCSD n'a pas été beaucoup utilisé pour le développement de programmes commerciaux pour l'Apple II – a priori la série Wizardry (« Sorcellerie » en édition française) est le seul programme connu écrit avec ce langage – il a en revanche été beaucoup utilisé par les étudiants… Jusqu'à l'avènement du célèbre Turbo Pascal ! Et le langage Pascal via d'autres implémentations est devenu le langage principal de la marque à la pomme sur ses ordinateurs suivants – Apple IIgs, Macintosh, Lisa – utilisé par exemple pour l'écriture de MacWrite ou la définition de l'API graphique des anciens Macintosh, QuickDraw. •

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel, Y. Grandmontagne
Nos experts techniques : D. Baeli, D. Djordjevic, W. Klein, S. Ollivier, I. Leontief, V. Thavonekham, C. Lakech, R. Linsolas, B. Holmes, C. Bitton, A. Zanchetta, N. Telera, S. Jeyakumaran, G. Bouveret, J. Grenat, V. Piard, W. Moulin, T. Gx, C. Villeneuve, A. Canut, M. Roussel, M. Krief
Couverture : © NASA et © Commitstrip - **Maquette :** Pierre Sandré.
Publicité : PC Presse, Tél.: 01 74 70 16 30, Fax : 01 40 90 70 81 - pub@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34
Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com - **Publicité :** benoit.gagnaire@programmez.com - Evenements / agenda : redaction@programmez.com
Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, septembre 2017

Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.
PDF : 35 € (monde entier) souscription sur www.programmez.com

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

WINDEV® Tech Tour 22

MERCI !

Merci à vous

Grâce à vous, le roadshow WTT (WINDEV Tech Tour) est la manifestation francophone professionnelle consacrée au développement la plus fréquentée au monde.
C'est toujours un grand plaisir de vous rencontrer.
Encore merci, et à très bientôt !

WWW.PCSOFT.FR